HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

MIT INNOVATION AG
Ein Unternehmen der MIT-GROUP

# MeteoCheck Relaunch mit Xamarin

# Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrsemester 2014


| | |
|---|---|
| Autor: | Patrick Pulfer |
| Betreuer: | Hansjörg Huser |
| Projektpartner: | MIT-Group |

# Inhaltsverzeichnis

# Erklärung

Ich erkläre hiermit,

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

Ort, Datum: Zürich, 29.05.2913

Name, Unterschrift:

Patrick Pulfer

# 1. Abstract

Diese Studienarbeit befasst sich mit der plattformübergreifenden Entwicklung für mobile Geräte mit Xamarin als Entwicklungsumgebung. Hinter Xamarin steht das Mono Framework, eine Open Source Implementierung des Microsofts .NET Framework, welches im Jahr 2001 erstmals veröffentlicht wurde.
Der Vorteil von Xamarin ist, dass native Applikationen für alle drei führenden mobilen Plattformen mit der gleichen Programmiersprache C# entwickelt werden können. Xamarin hat die Schnittstelle von Android und iOS mit regulären C# Klassen gekapselt.

Entwickler profitieren von der modernen Programmiersprache C# und deren Funktionalitäten wie LINQ, funktionalen Lambda Ausdrücken und Generics.

Im Gegensatz zu Web Apps wird das User Interface pro Plattform separat programmiert. Dadurch wird das gewünschte Benutzererlebnis erreicht und es können auch leistungskritische Anwendungen wie Spiele programmiert werden.

Als primäres Ziel gilt das gemeinsame Nutzen von Code. Mit Hilfe von portablen Klassenbibliotheken ist es möglich, den gleichen Code für alle drei Plattformen zu nutzen. Eine saubere Trennung zwischen dem User Interface und dem Business Layer ist erstrebenswert. Um dies zu erreichen, muss das Model-View-ViewModel Muster (MVVM) plattformübergreifend genutzt werden können. Das Framework MvvmCross stellt hierfür einige Bibliotheken zur Verfügung.

Nachdem die Grundlagen von Xamarin und MvvmCross bekannt waren, stand die Portierung der bestehenden Windows Phone App „MeteoCheck.ch“ auf Android und iOS im Vordergrund. Die Applikation bietet nebst den aktuellen Wetterprognosen auch zusätzlich Informationen wie zum Beispiel Lawineninformationen, Sonnenstand und Live Kamera Bilder.

Nach einer gründlichen Analyse wurde das vorhandene Projekt in eine portable Klassenbibliothek und in ein Windows Phone Projekt aufgeteilt. Die portable Klassenbibliothek enthält alle ViewModels und ist für die Kommunikation mit dem Web Server zuständig. MvvmCross hat zudem einen IoC Container und ein Navigation Controller implementiert, wodurch die Navigation der einzelnen Ansichten über die ViewModels gesteuert werden kann.

Die grösste Schwierigkeit bestand fortan bei der Entwicklung der einzelnen User Interfaces. Bei der Android Entwicklung war dies vergleichsweise einfach, da Java Implementierungen fast identisch mit C# abgebildet werden konnten. Bei iOS Applikationen mit der nativen Programmiersprache Objectiv-C, war dies durch den gesetzten zeitlichen Rahmen schwieriger, was für den weiteren Verlauf bedeutete, dass der Android Prototyp erweitert wurde.

Abschliessend kann gesagt werden, dass mit Xamarin gute Applikationen entwickelt werden können, sofern die notwendigen Kenntnisse über die Architektur und den Lebenszyklus der jeweiligen Plattformen vorhanden sind.

# 2. Aufgabenstellung

## Aufgabe

**Einarbeitung Xamarin**

- Xamarin Plattform Funktionalität
- Xamarin IDE versus Visual Studio
- UI Design Tools
- Component – Library
- Entwicklungsprozess, Cross Debugging
- SW Architekturen
    - Gemeinsame Funktionalität
    - Plattformspezifische Funktionalität (z.B. MVVM oder MVP)

**Relaunch Meteo Check**

- Ueberprüfung der Anforderungen
- Redesign UI (usability verbessern)
- Implementation basierend auf Xamarin:
- Zielplattformen: Windows 8 Mobile, Android, iOS

**Resultate**:

- Erfahrungsbericht Xamarin
- Lauffähige Software
- Dokumentation gemäss HSR-Vorgaben

### Auftraggeber

MIT-Group, Freienbach

### Projektteam

Patrick Pulfer

### Betreuung HSR

Hansjörg Huser

Management Summary

## Ausgangslage

Xamarin ist als Entwicklungsumgebung gesetzt. Bereits vorhanden ist die Windows Phone App „MeteoCheck.ch“. Ziel ist es, diese Applikation auf die anderen zwei führenden Plattformen zu portieren.

Dies sind Android und iOS. Von Xamarin werden beide Plattformen unterstützt. Da die Windows Phone Entwicklung als native Sprache C# verwendet, steht die Portierung auf Android und iOS im Vordergrund.

In der folgenden Tabelle sieht man die weltweiten Smartphone Verkäufe an Endbenutzer.

| Operating System | 2013 Units | 2013 Market Share (%) | 2012 Units | 2012 Market Share (%) |
|---|---|---|---|---|
| Android | 758,719.9 | 78.4 | 451,621.0 | 66.4 |
| iOS | 150,785.9 | 15.6 | 130,133.2 | 19.1 |
| Microsoft | 30,842.9 | 3.2 | 16,940.7 | 2.5 |
| BlackBerry | 18,605.9 | 1.9 | 34,210.3 | 5.0 |
| Other OS | 8,821.2 | 0.9 | 47,203.0 | 6.9 |
| **Total** | **967,775.8** | **100.0** | **680,108.2** | **100.0** |

*Quelle: Gartner (Februar 2014) [1]* 1000 Einheiten

Android und iOS haben zusammen einen Marktanteil von über 90%. Deswegen ist für Unternehmen wichtig, ihre Mobile Applikation auch auf diesen Geräten zu veröffentlichen.

Um dies effizient zu erreichen, wird in diesem Projekt Xamarin benutzt. Es sollen vor allem Erfahrungen im Bereich der Cross Plattform Entwicklung mit Xamarin gesammelt werden, um für zukünftige Projekte gerüstet zu sein.

## MeteoCheck.ch Windows Phone Applikation


Die MeteoCheck.ch App ist eine Applikation, mit der man die Schweizer Wetterprognosen von über 150 Tourismusregionen abrufen kann.

Nebst dieser Hauptfunktionalität können folgende Infografiken über die App aufgerufen werden:

- Lawinenbulletin
- Sonnenstand
- Niederschlagsradar
- Grillwetter
- Prognosen der Vergangenheit

Beliebte Orte können als Favoriten markiert werden. Es gibt keine Benutzer Anmeldung, noch werden sonstige Daten des Benutzers gespeichert. Benutzer Einstellungen, wie Favoriten, sind nur lokal auf dem Gerät vorhanden.

Das User Interface ist mit dem Panorama Control umgesetzt. Dieses typische Windows Phone User Control gliedert die verschiedenen Ansichten horizontal nebeneinander.



Die Regionen sind hier führend bei der Navigation. Aktuell ist es nicht möglich, direkt einen Ort zu suchen. Will man zu einem bestimmten Ort die Wetterprognosen anschauen, muss man zuerst wissen, zu welcher Region dieser Ort gehört.

## Vorgehen, Technologien

Dier Arbeit ist gemäss dem Projektplan umzusetzen. Die Meilensteine können dem Projektplan entnommen werden. Folgend wird kurz der geplante Ablauf beschrieben.

1. **Xamarin Einarbeiten**
   Anfangs müssen die notwendigen Kenntnisse in der plattformübergreifenden Entwicklung mit Xamarin angeeignet werden. Für die iOS Entwicklung wird eine virtuelle Maschine mit OS X Mavericks benötigt, da iOS Applikationen nur auf einem Apple Computer kompiliert werden können.

2. **Analyse Windows Phone Applikation**
   Was für Funktionalitäten bietet die App? Wie ist dies momentan umgesetzt?
   Wie findet die Kommunikation mit dem Server statt? Was für mobile Services, z.B GPS, werden von der Windows Phone Applikation genutzt?

3. **Refactoring Windows Phone Applikation, Core Library erstellen**
   Bevor die Portierung auf die anderen Plattformen umgesetzt wird, muss das aktuelle Windows Phone Projekt an einigen Stellen überarbeitet werden.
   Da die gemeinsame Nutzung von Code im Vordergrund steht, wird eine saubere Trennung des Applikations-Layer und des Business-Layer angestrebt. Dies erfordert die Einführung einer Core Library, welche von allen drei Plattformen genutzt werden kann.

4. **Android Prototyp**
   Der Android Prototyp wird als erstes entwickelt. Native Android Applikationen sind mit Java programmiert, dessen Syntax C# sehr ähnlich ist. Im Vordergrund stehen die Kommunikation mit dem Server und die Nutzung der gleichen Core Library.

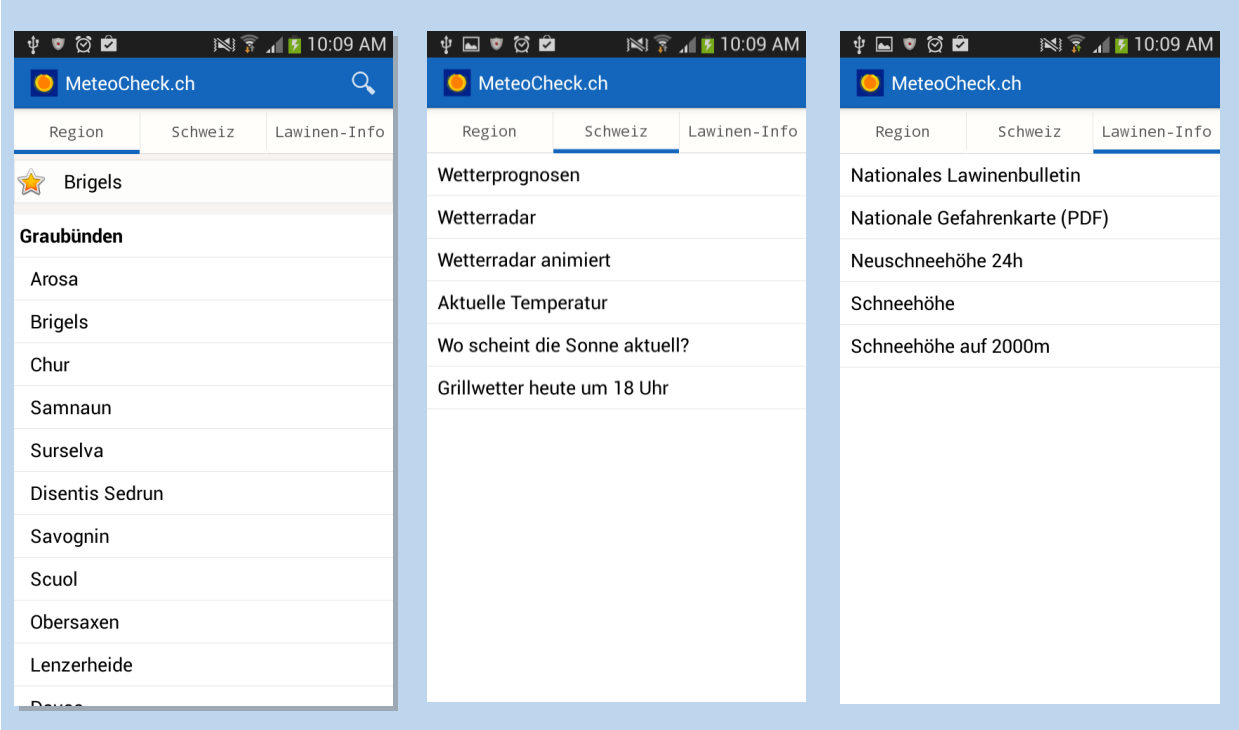
5. **iPhone Prototyp**
   Der iPhone Prototyp soll die gleiche Core Library nutzen. Die Aufwandschätzung für die Entwicklung ist deutlich schwieriger, da keine Vorkenntnisse in der OS X Entwicklung vorhanden sind.

6. **Final Release, Erfahrungsbericht**
   Nachdem die beiden Prototypen entwickelt worden sind, soll die Android App mit möglichst viel Funktionalität der bestehenden Windows Phone App erweitert werden. Die gewonnenen Erfahrungen sollen niedergeschrieben werden.

## Resultat

### Android

Die bestehende MeteoCheck.ch App ist mit dem Panorama Control für Windows Phone Applikationen umgesetzt. Für Android und iOS gibt es dies nicht. Um ein ähnliches Benutzererlebnis auf den anderen Plattformen zu bieten, wurde mit einer Tab Navigation gearbeitet.

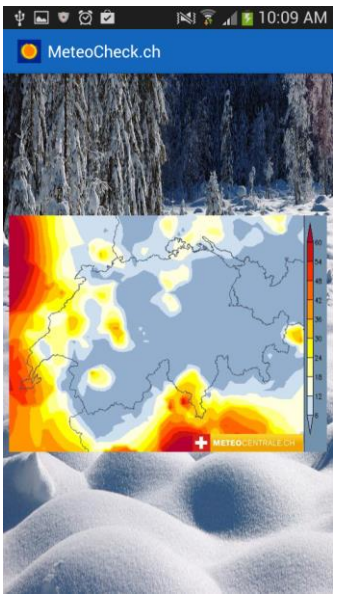



Im Vergleich zur Windows Phone App wurde die Region Ansicht um die dazugehörigen Orte erweitert. So ist es möglich direkt auf den Ort zu gelangen. Favoriten sind mit einem Stern markiert und werden oberhalb der ListView angezeigt. Eine Suche wurde zusätzlich implementiert.

Per Klick auf die Orte erhält man die aktuellen Wetter Daten.

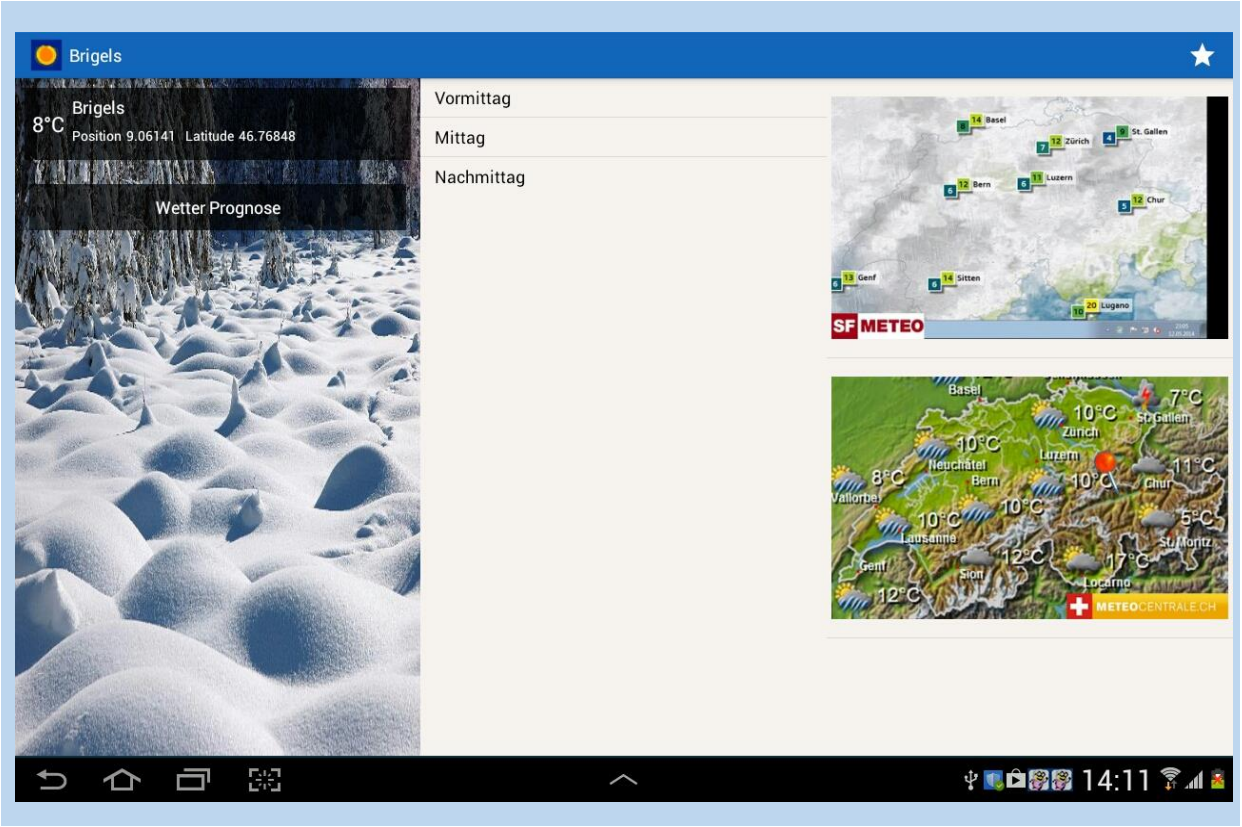
Die Ansicht der Schweiz und Lawinen-Infos wurde gemäss der Windows Phone App umgesetzt. Klickt man auf ein Item der Schweiz Liste, wird die dazugehörige Infografik geladen.

Bei der Lawinen-Info Liste wird der Browser auf dem Phone mit der dazugehörigen Url gestartet. Dies soll auch aufzeigen, wie man den Browser auf den verschiedenen Plattformen aufrufen kann.



Die Detail Ansicht der einzelnen Orte ist für unterschiedliche Auflösungen optimiert. Dies war ein Wunsch seitens des Auftragsgebers, um auch in dieser Hinsicht Erfahrungen zu sammeln.

Die Ansicht ist in drei Fragmente unterteilt. Bedient man die Applikation mit dem Handy, dann wird jedes Fragment einzeln dargestellt. Auf dem Tablet können die Fragmente nebeneinander aufgelistet werden.




Über den Stern oben rechts kann der gewählte Ort den Favoriten hinzugefügt werden und erscheint fortan auf der Hauptseite.

## Fehlende Funktionen

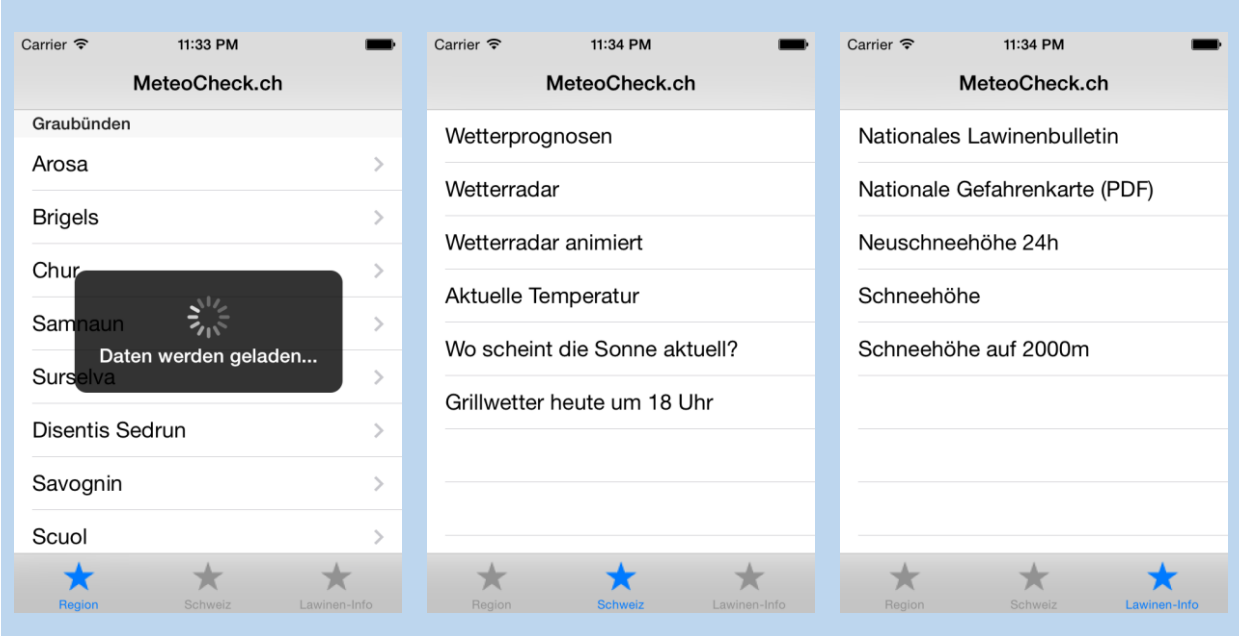
Aus zeitlichen Gründen kann nicht die gesamte Funktionalität umgesetzt werden. So fehlen im Gegensatz zur Windows Phone Applikation die aktuellen Live Kamera Bilder und die Möglichkeit vergangene Infografiken zu laden.

### iPhone Prototyp

Der iPhone Prototyp stellte sich als wesentlich schwieriger heraus. Die ganze Vorarbeit der Portable Class Library und die bereits optimierten ViewModels standen zwar zur Verfügung, jedoch machte sich das fehlende Knowhow in der iOS Entwicklung schnell bemerkbar. Insgesamt konnte für den iPhone Prototyp nur 32 Stunden eingeplant werden, was Entwicklung, Wissensbeschaffung und Dokumentation beinhaltete.

Die Abhängigkeit zu OS X als Betriebssystem zum Kompilieren, erschwerte zudem die Entwicklung und wird im technischen Bericht genauer erläutert.

Der Prototyp beschränkt sich daher nur auf die drei Ansichten der Hauptnavigation.




## Fehlende Funktionen

Bei der iOS Version handelt es sich nur um einen Prototypen. Jegliche Funktionalitäten zum Anzeigen von Infografiken und weiteren Detailinformationen fehlen.

## Ausblick

In der Hälfte meiner Studienarbeit begann die MIT Innovation AG  eine weitere Windows Phone Applikation auf Android zu portieren. Meine gewonnenen Erfahrungen konnten bereits für dieses Projekt genutzt werden. Vor allem der Einsatz des MvvmCross Framework von Beginn an, erleichterte die Arbeit enorm.

Der aktuelle Stand der Android App ist noch nicht Release fähig. Die App muss noch um weitere Funktionalitäten ergänzt werden, um im Android Play Store veröffentlicht werden zu können.

Auch im Bereich Design ist noch ein Experte beizuziehen. Es hat sich während der Arbeit gezeigt, dass es enorm aufwändig ist, ein spezielles Design zu gestalten. Weicht man nur von den Standard Komponenten ab oder will dem Benutzer eine speziellere Animation bringen, kostet dies enorm viel Zeit.

# 3. Mobile Entwicklung Allgemein

## Einführung

Spätestens seit Apple das iPhone lanciert hat, sind die Smartphones für viele Personen ein ständiger Begleiter. Schaut man heute in den öffentlichen Verkehrsmitteln in ein Buch, gehört man zur Minderheit. Das Smartphone prägt unseren Tagesablauf.

Folglich ist es für Software Dienstleister schon lange unumgänglich, sich mit der Entwicklung für mobile Geräte auseinanderzusetzen.

In diesem Abschnitt wird kurz auf die heute möglichen Varianten eingegangen, wie man Mobile Applikation entwickeln kann.

Allgemein gilt, dass keine Variante eine universale Lösung für alle Probleme bietet. Es ist immer abhängig von Faktoren wie: den Anforderungen einer Applikation, Entwickler Fähigkeiten und der vorhandenen Zeit.

## Native Entwicklung

Jede Plattform hat ihre eigene native Programmiersprache.

| Plattform | Native Programmiersprache | Marktanteil 2013 |
|---|---|---|
| **Android** | Java | 78.4 % |
| **Apple iOS** | Objectiv-C | 15.6 % |
| **Windows Phone** | C# | 3.2 % |

Native Applikationen sind Programme, welche direkt auf der Betriebssystem-Ebene laufen. Schnittstellen können direkt angesprochen und die Hardware Funktionalität somit zu 100% genutzt werden. Durch diese optimale Ausnutzung kann ein sehr gutes Benutzererlebnis erreicht werden. Für rechenintensive und komplexe Apps, wie zum Beispiel Spiele ist die native App sicher die richtige Wahl.

Der grosse Nachteil einer nativen Applikation ist, dass der geschriebene Code nicht einfach auf eine andere Plattform adaptiert werden kann. Die Applikation muss daher pro Plattform entwickelt werden, was Zeit und Knowhow benötigt.

Native Apps können über den App Store vertrieben werden und erreichen dadurch ein grosses Publikum.

## WebApp Entwicklung

Bei der WebApp Entwicklung ist man unabhängiger vom Gerät. WebApps sind nichts anderes als Webseiten, welche für die mobilen Endgeräte spezifiziert sind.
Mit HTML5 und CSS3 hat man bereits einige Funktionalitäten, wie Offline Data Storage, Animation, Caching und Touch. Jedoch ist hier meist das Benutzererlebnis schlechter, da sich die Applikation oft träge anfühlt. Zudem kann nicht auf die Hardware Komponenten zugegriffen werden.

## Hybrid Entwicklung

Hybrid Apps sind eine Kombination der oben zwei erwähnten Architekturen. Hybrid Apps werden auch in HTML5 programmiert und mit Hilfe eines  Containers zu einer mobilen Architektur verbunden. Der weitaus bekannteste Anbieter ist dabei PhoneGap. Mittels PhoneGap kann man viele Hardware Funktionalitäten über Javascript ansprechen.

Im Vergleich zu native Apps benötigen Hybrid Apps relativ viel Arbeitsspeicher. Viele grafische Elemente und Animationen können die Applikation sehr verlangsamen.

Stellt sich nun die Frage, ob es eine Alternative zu diesen drei Varianten gibt, welche doch ein befriedigendes Benutzererlebnis bieten und den Aufwand und die Kosten im Rahmen halten.

Hier kommt Xamarin zum Zug.

# 4. Xamarin

## Geschichte

Hinter dem Unternehmen Xamarin steht das Mono Framework. Schon lange ist Mono ein Begriff in der IT und steht für die Open Source Implementation des Microsoft .Net Framework basierend auf dem ECMA Standard für C# und der Common Language Runtime.

Xamarin wurde erst im Jahre 2011 gegründet und hat bereits über 500‘000 Entwickler für sich gewonnen. Dies liegt zum einen an der Marketing Strategie, wie auch an der Partnerschaft mit Microsoft und der vollen Integration in Visual Studio.
Ihre Webseite ist sehr übersichtlich und einfach gegliedert. Es sind enorm viele Beispiele vorhanden und man hat schnell das Gefühl am richtigen Ort zu sein, wenn man sich mit Cross-Plattform Entwicklung auseinandersetzt.

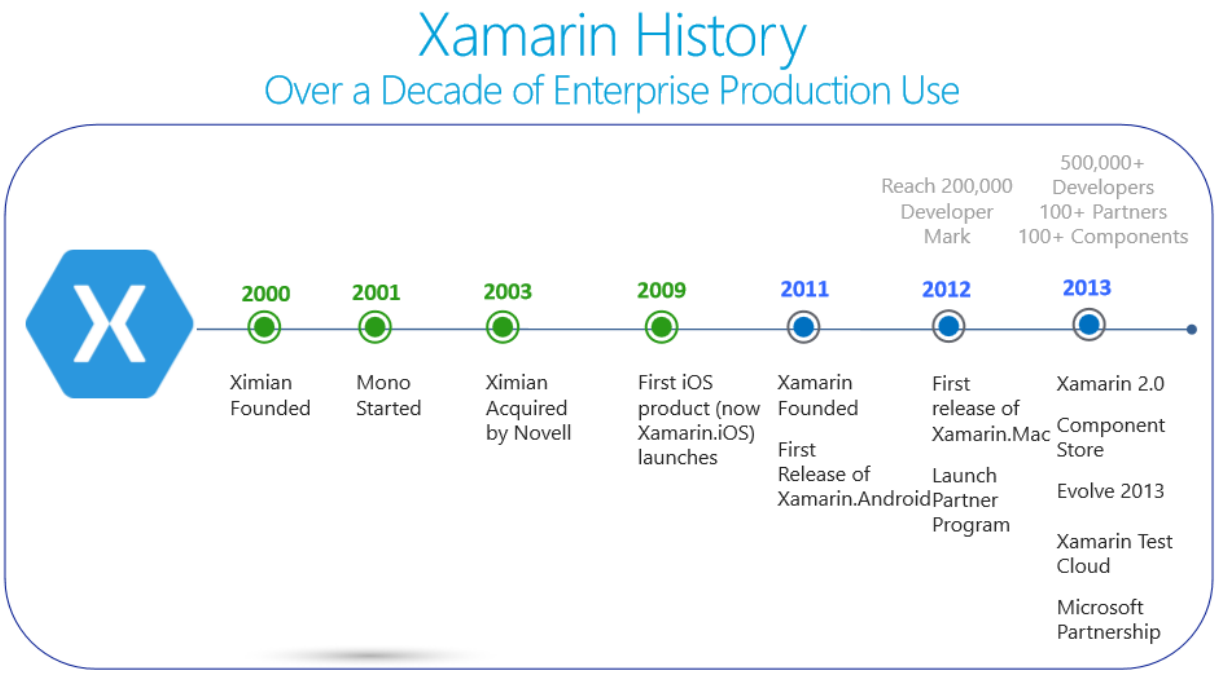



*Abbildung 1: xamarin.com*

Ximian startete 2001 ursprünglich das Projekt Mono unter der Führung von Miguel de Icaza, welcher auch Gründer von Xamarin ist.

## Xamarin Funktionalität

Xamarin ist die einzige Entwicklungsumgebung auf welcher man native Applikation in der gleichen Programmiersprache (C#) für die drei führenden mobilen Plattformen entwickeln kann. Die Windows Phone Entwicklung ist dabei mit C# als native Sprache bereits abgedeckt.

Mit C# erhält man einige Vorteile gegenüber Objectiv-C und Java. Dies sind unter anderem LINQ für den Zugriff auf Daten, Funktionale Konstrukte mit Lambdas oder die Unterstützung für Parallele Programmierung.

Xamarin hat eine Binding Technologie entwickelt, durch die alle nativen APIs angesprochen werden können. Alle Hardware Funktionalitäten sind durch reguläre C# Klassen abgebildet und somit typsicher und einfach zu gebrauchen.

Bestehende native Implementierungen mit Objectiv-C oder Java können wiederverwendet werden. Xamarin bietet eine einfache deklarativen Binding-Syntax an, solche Komponenten einzubinden. Somit können auch Komponenten von Drittanbietern verwendet werden.

Je nach Plattform erstellt der Xamarin Cross-Compiler einen unterschiedlichen Output.


*Abbildung 2: xamarin.com*

Apple verbietet die dynamische Code Generierung auf den mobilen Geräten. Daher wird vor dem Verteilen, der Code mit dem Ahead-Of-Time (AOT) Compiler in ein ARM Binary umgewandelt.


*Abbildung 3: xarmarin.com*

Xamarion.Android nutzt den Just-In-Time (JIT) Compiler auf dem Android Device. Einige Funktionen werden über Mono direkt vom Linux Kernel gestartet. Die meisten System Funktionalitäten wie Audio, Grafik, OpenGL etc. sind nur über die Dalvik Java API erreichbar und werden mit Hilfe von Java Native Interface (JNI) aufgerufen.

## Xamarin Studio IDE vs. Xamarin für Visual Studio

Bei der Entwicklung von Xamarin Projekten hat man zwei Möglichkeiten. Entweder entwickelt man im Visual Studio oder mit der Xamarin Studio IDE.

Welche Variante von Entwicklern bevorzugt wird, ist oft sehr unterschiedlich. Die Xamarin Studio IDE ist im Vergleich zum Visual Studio sehr schlank und für die Cross Plattform Entwicklung ausgerichtet. Zusätzlich ist es nicht möglich Visual Studio auf einem Mac zu installieren, wodurch man gezwungen wird mit dem Xamarin Studio zu arbeiten.

Arbeitet man hingegen auf einem Windows Computer und ist mit dem Visual Studio vertraut, wird diese Variante oft bevorzugt. Neben der gewohnten Umgebung kann man auch von Erweiterungen wie ReSharper profitieren, was einem die Entwicklung enorm erleichtert.

Aus diesem Grund wurde die Visual Studio Variante in diesem Projekt gewählt.

## Cross Plattform Entwicklung mit Xamarin

Auf der Webseite von Xamarin wird behauptet, dass bis zu 90% des Source Code auf allen Plattformen gemeinsam genutzt werden kann. Diese Zahlen sind natürlich mit Vorsicht zu geniessen. Tatsächlich kann jedoch mit der richtigen Architektur und gewissen zusätzlichen Libraries, wie von MvvmCross, viel Code in die Core Library verschoben werden.


*Abbildung 4: xamarin.com [2]*

Das User Interface muss auf allen Plattformen separat entwickelt werden. Dies heisst, dass man trotz der Möglichkeit C# zu benutzen, die Architektur und Design Vorgaben der einzelnen Plattformen gut kennen muss, um erfolgreich Applikation zu entwickeln.

Um den Code gemeinsam zu nutzen gibt es  zwei bevorzugte Möglichkeiten, welche im nächsten Abschnitt genau erklärt werden.

## File Linking

Bei der Variante File Linking wird für jede Plattform ein C#-Projekt erstellt. Der gemeinsame Code wird dann via File-Links hinzugefügt. Wenn nun der Code verändert wird, hat dies Auswirkungen auf alle Projekte.

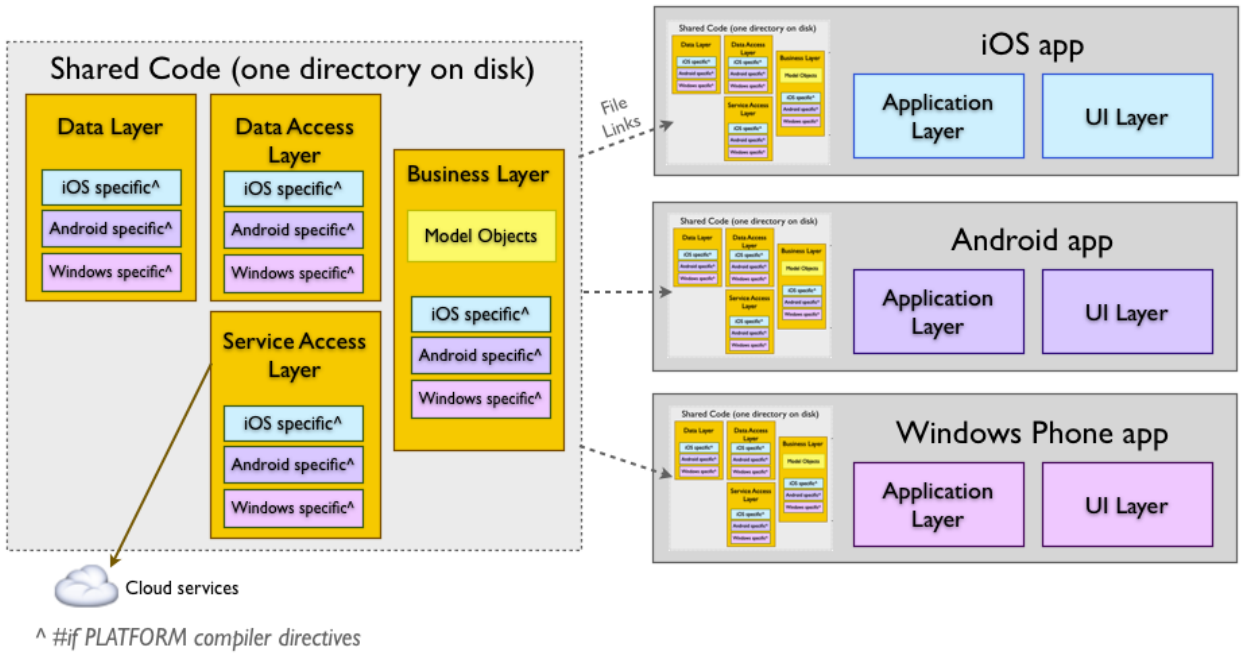



*Abbildung 5: xamarin.com [5]*

Ein Vorteil dieser Variante ist, dass mit #if- Compiler-Direktiven in den einzelnen gemeinsamen Code Fragmenten plattformspezifische Anweisungen platziert werden können. Im Unterschied zur Portable Class Library kann dadurch auf plattformspezifische Assemblies referenziert werden.

Dies bringt jedoch auch den Nachteil, dass jede Änderung in allen Projekten nachgezogen werden muss. Wenn zum Beispiel eine Klasse hinzugefügt wird, muss in allen Projekten das File neu gelinkt werden.

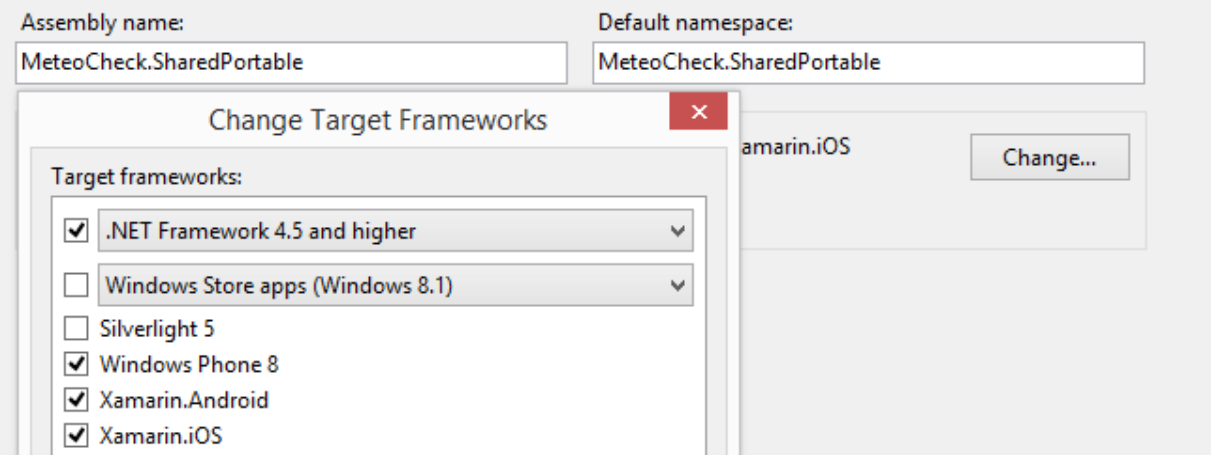
Der mühsamste Effekt dieser Variante ist die Problematik beim Refactoring. Daher dass die Projekte unabhängig sind, funktioniert das Refactoring jeweils nur für das geladene Projekt.

Aufgrund dieser Nachteile wurde dann auch die zweite Variante mit Hilfe einer Portable Class Library bevorzugt.

## Portable Class Library (PCL)

Portable Klassenbibliotheken ermöglichen es, gemeinsamen Code für unterschiedliche Plattformen zu nutzen. Sie unterstützen eine Teilmenge der Assemblies des .NET Framework, Windows Store apps, Silverlight, Windows Phone und falls Xamarin installiert ist auch Xamarin.Android und Xamarin.IOs.

Unter den Projekt Eigenschaften können die gewünschten Plattformen ausgewählt werden.




Über diese Auswahlen kann nun Visual Studio erkennen, welche .NET Frameworks und Namespaces in der Portable Class Library benutzt werden können.
Je nach Konfiguration hat man mehr oder weniger Funktionalitäten des .NET Frameworks zur Verfügung. Xamarin unterstützt wie man in der unten stehenden Tabelle sieht alle wichtigen Funktionalitäten.

| Funktion | .Net Framwork | Windows Phone | Xamarin |
|---|---|---|---|
| **Core** | Yes | Yes | Yes |
| **LINQ** | Yes | Yes | Yes |
| **IQueryable** | Yes | Yes | Yes |
| **Serialization** | Yes | Yes | Yes |
| **Data Annotations** | Yes | Yes | Yes |

Will man es nun genauer wissen, ob eine bestimmte Klasse in einer PCL genutzt werden kann, findet man dies unter der Version Information der jeweiligen Klasse auf MSDN.

### Version Information

**.NET Framework**
Supported in: 4.5, 4, 3.5
**.NET Framework Client Profile**
Supported in: 4, 3.5 SP1
**Portable Class Library**
Supported in: Portable Class Library
**.NET for Windows Store apps**
Supported in: Windows 8

Häufig will man in einer PCL auch Methoden aufrufen, welche plattformabhängig unterschiedliche Implementationen haben. Mit Hilfe des Provider Pattern oder Dependency Injection können plattformspezifische Funktionalitäten übergeben werden.

Folgendes Beispiel von der MSDN Seite [3] zeigt, wie dies mit Hilfe einer abstrakten Klasse gemacht werden kann.

```
namespace ExamplePortableLibrary
{
    public abstract class ExampleLocalSettings
    {
        public abstract void SetLocalValue(string name, object value);
        public static ExampleLocalSettings Instance { get; set; }
    }
}
```

Aufruf aus der PCL:

```
ExampleLocalSettings.Instance.SetLocalValue("ExampleSetting", "New value to add");
```

Plattformspezifische Implementation :

```
namespace SilverlightApplication1
{
    class SilverlightImplementation : ExampleLocalSettings
    {
        public override void SetLocalValue(string name, object value)
        {
            IsolatedStorageSettings.ApplicationSettings.Add(name, value);
        }
    }
}
```

Damit dies auch funktioniert wird meistens beim Starten der Applikation die Instanz erzeugt und dem Property zugewiesen:

```
ExampleLocalSettings.Instance = new SilverlightImplementation();
```

Wie dies mit Hilfe eines IoC Container noch eleganter gelöst werden kann, ist im Kapitel MvvmCross beschrieben.

# 5. MvvmCross

## Einführung

Nachdem man sich eine Weile mit Xamarin auseinandergesetzt hat, gelangt man schnell zu dem Framework MvvmCross. Wie der Name sagt, hat MvvmCross sich zum Ziel gesetzt, das MVVM Pattern plattformübergreifend zur Verfügung zu stellen.

## MVVM (Model – View – ViewModel) Pattern

Ziel des MVVM Pattern ist die klare Trennung der ViewModels vom User Interface. Wird das MVVM Pattern richtig eingesetzt, kann viel duplizierter Code gespart werden.

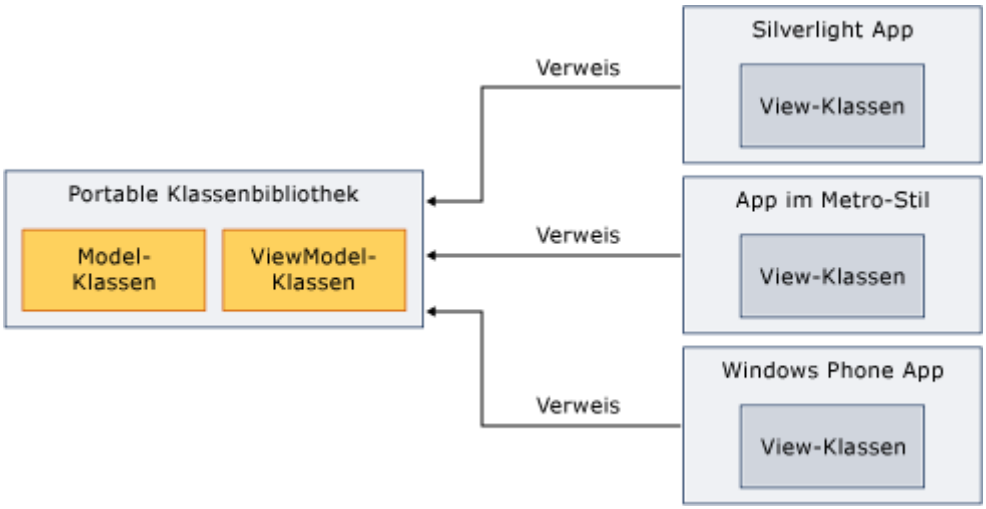



*Abbildung 6: MSDN [6]*

## MVVM vs. MVP

Beide Design Muster haben als Ziel eine klare Trennung zwischen Model, Ansicht und Präsentationslogik (Separation of Concern). MVVM ist eine Spezialisierung von MVP.

Beide verschieben das Verhalten der View in ein separates User Interface Model (Presenter/ ViewModel).


Der grösste Unterschied ist die Beziehung zwischen View und ViewModel/Presenter.

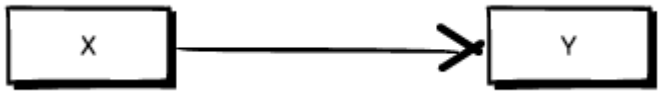
Das MVVM Pattern kann nur dann eingesetzt werden, wenn die Views einen DataContext unterstützen. Das bekannteste Grafikframework, welches MVVM voraussetzt ist WPF.
Mit Hilfe von MVVMCross kann nun auch plattformübergreifend dieses Pattern eingesetzt werden.

## IoC (Inversion of Control) Container

Wie bereits im Kapitel Portable Class Library angedeutet, verwendet MvvmCross einen IoC Container. Der Vorteil eines IoC Container ist die lose Kopplung von Komponenten. Dies ist in unserem Fall sehr wichtig, da wir viele unterschiedliche Implementationen pro Plattform haben und diese wenn möglich über unsere Portable Class Library ansteuern wollen.

Das Pattern welches meistens hierfür verwendet wird nennt sich „Dependency Injection“.

Beispiel Dependency Injection:




Klasse X konsumiert Klasse Y.

Anstatt hier eine Abhängigkeit zur Klasse Y zu schaffen, würde es reichen, dass wir in der Klasse X nur das Verhalten (Interface) von Y kennen. Somit erreichen wir eine lose Kopplung dieser Komponenten und können für Y beliebige unterschiedliche Implementierungen haben, solange diese das gleiche Interface aufweisen.

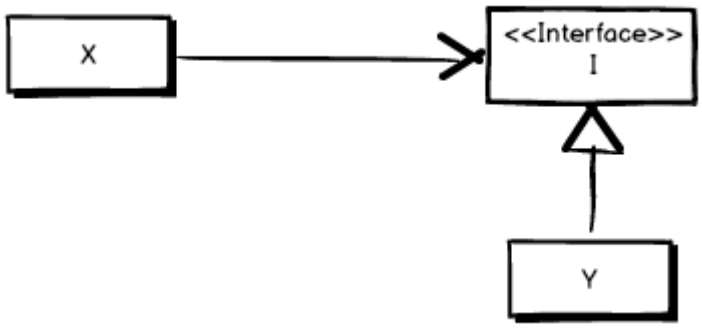



*Abbildung 7+8: Beispiel von Joel Abrahamsson [4]*

In der plattformübergreifenden Entwicklung spielt dies eine zentrale Rolle, da wir je nach Plattform das Verhalten unterschiedlich implementieren müssen.

Hier ein Beispiel von Dependency Injection im aktuellen MeteoCheck.ch Projekt:

```
public class LocationViewModel : ViewModelBase
{
    private readonly ILocationService _locationService;

    public LocationViewModel(ILocationService locationService)
    {
        _locationService = locationService;
    }

}
```

Der LocationService ermittelt die GPS Koordinaten.

## MvvmCross Service Locator

MvvmCross macht auch Gebrauch von dem Service Locator Pattern.

MvvmCross stellt eine statische Klasse „Mvx“ zur Verfügung, welche für das Registrieren und Abfragen von Interfaces und deren Implementation verantwortlich ist.

Mögliche Registration:

```
Mvx.RegisterSingleton<IAppData>(new AppData());
```

Von überall kann nun mit folgendem Befehl dieselbe Instanz empfangen werden:

```
var AppData = Mvx.Resolve<IAppData>();
```

## MvvmCross Plugins

MvvmCross stellt bereits einige wichtige Plugins zur Verfügung, welche plattformübergreifend genutzt werden können.

Hier ein paar wichtige Plugins, welche von allen Plattformen unterstützt werden:

- Email
- File
- Location
- Messenger
- PhoneCall
- Sqlite
- WebBrowser

Mit Hilfe dieser bereits vorhandenen Plugins, kann nun von der Portable Class Library zum Beispiel ein Anruf gestartet werden oder die aktuelle Position ermittelt werden.

Diese Plugins werden laufend ergänzt und falls eine gewünschte Funktionalität noch nicht vorhanden ist, gibt es auf GitHub eine gute Anleitung wie man ein eigenes Plugin entwickeln kann. [7]

# 6. Portierung der MeteoCheck.ch App

## Einführung

Um die bestehende Windows Phone Applikation portieren zu können, musste zuerst das bestehende Projekt analysiert werden. Wie in den vorangegangen Kapitel beschrieben, ist es von zentraler Bedeutung möglichst viel Code gemeinsam nutzen zu können.

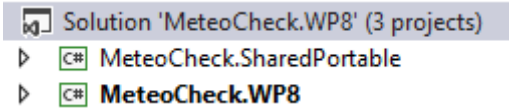
## Refactoring MeteoCheck.ch WP8 für Cross-Plattform Entwicklung

Die existierende Windows Phone Applikation bestand anfangs nur aus einem Projekt. Es fehlte die klare Trennung des Applikations-Layer und des Business-Layer auf Projekt Ebene.

Es stand also zuerst ein Refactoring der Windows Phone Applikation an, bevor das User Interface für die Android Entwicklung angegangen werden konnte.

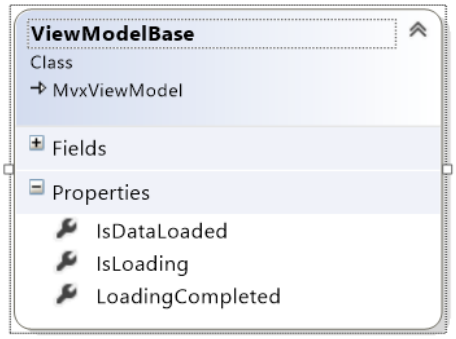
### Shared Library (Core)

Aller erstens wurde eine Portable Class Library Namens „MeteoCheck.SharedPortable“ erstellt.




In die PCL wurden nun mit Hilfe von Resharper alle ViewModels verschoben. Die Verwendung von Reshaper sei hier speziell zu erwähen, da ohne Resharper ein so grosses Refactoring kaum möglich gewesen wäre.

In den vorhanden ViewModels war sehr viel duplizierter Code vorhanden. Daher wurde eine abstrakte ViewModelBase Klasse erstellt.




Damit das MVVM Pattern plattformübergreifend genutzt werden kann, muss diese ViewModelBase Klasse von MvxViewModel erben.

MvxViewModel ist eine Abstrakte Klasse, welche über die Libraries von MvvmCross hinzugefügt wird.
Dadurch kann nun die Navigation über die ViewModels gesteuert werden.

Über die Methode ShowViewModel<SliceInfosViewModel>() wird nun die zugehörige View gesucht. MvvmCross macht dies mit Hilfe des Namens und Reflection.
SliceInfosViewModel -> SliceInfosView wird gesucht.

```
private MvxCommand _showPrognoseCommand;
public ICommand ShowPrognoseCommand
{
   get {
     _showPrognoseCommand = _showPrognoseCommand ??
     new MvxCommand(() => ShowViewModel<SliceInfosViewModel>());
     return _showPrognoseCommand;
   }
}
```

## Async Await

Damit das User Interface nicht einfriert, wird für die Server Kommunikation mit asynchronen Aufrufen gearbeitet.

Alle Aufrufe gehen über die statische asynchrone Methode LoadStringFromUriAsync. Da die Daten vom Server komprimiert als GZip kommen, muss dem HttpClient zusätzlich die Decompression Methode mitgegeben werden.

```
public static async Task<string> LoadStringFromUriAsync(Uri uri)
{
       try
       {
         var handler = new HttpClientHandler();
         if (handler.SupportsAutomaticDecompression)
         {
           handler.AutomaticDecompression = DecompressionMethods.GZip |
                                            DecompressionMethods.Deflate;

         }
         using (var httpClient = new HttpClient(handler))
         {
             return await httpClient.GetStringAsync(uri);
         }
       }
       catch (Exception){ }
       return "";
}
```

Dies funktioniert für alle Plattformen.

# Android Applikation

## Einführung

Als erste Plattform für die Portierung wurde Android gewählt. Android verwendet als native Programmiersprache Java. Dass Java und C# sich ähneln half bei der Entwicklung enorm. Oft musste nur die Namenskonventionen von C# eingehalten werden, um Java Beispiele zu adaptieren.

## Infrastruktur

Damit Android Applikationen entwickelt werden können, braucht man zuerst einige Komponenten.

**Java JDK**

Android Applikationen sind in Java implementiert. Daher benötigt man das Java Development Kit.

**Android SDK**

Der Android Software Development Kit stellt die nötigen API Bibliotheken und Entwicklungstools zum Kompilieren, Testen und Debuggen von Android Applikationen zur Verfügung.

Zusätzlich bekommt man mit dem SDK auch ein Android Emulator. Leider ist der Android Emulator sehr langsam und braucht enorm viele Ressourcen. Es ist sehr zu empfehlen direkt auf einem Android Device zu testen.

**Xamarin Studio**

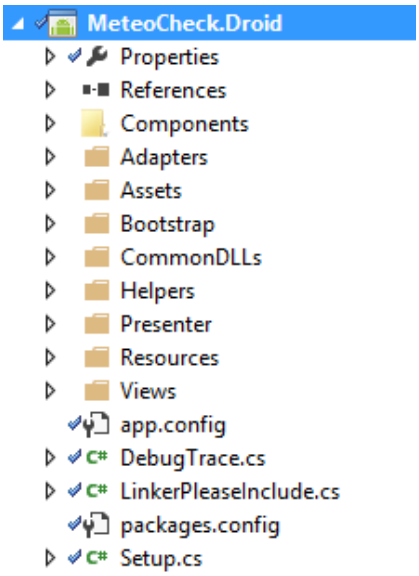
Zu guter Letzt wird das Xamarin Studio benötigt. Der Installer des Xamarin Studio prüft auch, ob die beiden anderen Komponenten bereits installiert sind und lädt diese gegebenenfalls noch herunter.

Leider funktionierte der Installer nicht wunschgemäss, so dass die einzelnen Komponenten manuell installiert werden mussten

Mit der Installation des Xamarin Studio kann nun entweder im Visual Studio direkt oder in der Xamarin Studio IDE gearbeitet werden. Für dieses Projekt wurde die Visual Studio Umgebung gewählt, da auf Erweiterungen wie Resharper nicht verzichtet werden wollte.

## Android Projekt Visual Studio

Durch die Installation von Xamarin hat man im Visual Studio ein neues Projekt Template für Android Applikationen.




### Projekt Struktur

#### Components

Jedes Xamarin Projekt enthält einen Ordner Namens Components. Xamarin bietet einen Online Component Store, in dem man einige kostenpflichtige und frei erhältliche Komponenten herunterladen kann.

Leider gibt es noch nicht sehr viele gute Komponenten. Die Einzige von mir verwendete Komponente, ist die Android Support Library, um ältere Versionen von Android zu unterstützen.

#### Assets

Der Ordner Assets ist für Ressourcen gedacht, welche direkt von der Android Applikation gelesen werden können.

#### Bootstrap

Bootstrap ist ein Ordner, welcher automatisch über die MvvmCross Plugins via NuGet erstellt wurde. Dieser enthält für jedes Plugin eine Bootstrap Klasse, welche beim Starten der Applikation via Reflection gesucht und dann instanziert wird.

Beispiel Location Plugin:

```
public class LocationPluginBootstrap
        : MvxPluginBootstrapAction<Cirrious.MvvmCross.Plugins.Location.PluginLoader>
    {
    }
```

#### CommonDLLs

In diesem Ordner sind zwei Assemblies enthalten, welche nicht über NuGet hinzugefügt werden können.

Dies ist erstens ein MonoDroidToolkit vom Entwickler James Montemagno, welcher einige Java Komponenten für Android bereitstellt und zweitens ein Assembly vom Entwickler Cheesbaron, mit dem die Tab Navigation umgesetzt wurde. Beide Entwickler erweitern fortlaufend Xamarin mit neuen User Interface Komponenten.

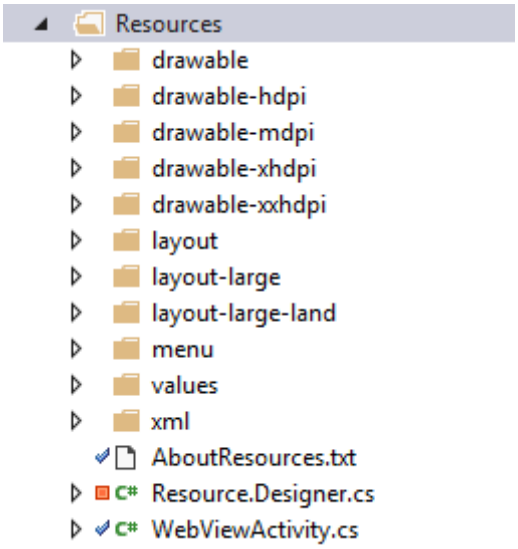
#### Helpers

Dieser Namensraum ist für eigene User Interface Erweiterungen gedacht.

#### Presenter

Enthält die Klasse CustomerPresenter, welche für die Navigation zwischen den Views und deren Animation verantwortlich ist.

**Resources**




In diesem Ordner sind alle Layout Definitionen für die Applikation abgelegt. Will man mehrere Auflösungen unterstützen, hat man die Möglichkeit mit verschiedenen Unterordner zu arbeiten.

Allgemein kann gesagt werden, dass in den Drawable Ordner alle Bilder liegen.

In den Layout Ordner sind die Views mittels XML Deklaration definiert. Diese können entweder von dem integrierten Designer oder per Standard XML Editor bearbeitet werden.

Die Resource.Designer Klasse wird jeweils beim Kompilieren erstellt.

**Views**

In diesem Ordner sind alle Activities und Fragments abgelegt. Eine Activity stellt ein User Interface bereit, mit welchem der Benutzer interagieren kann. Grundsätzlich könnte per Code das ganze User Interface gezeichnet werden. Jedoch wird meistens ein Layout File von dem Resourcen Ordner geladen, da es übersichtlicher ist die Layouts deklarativ per XML zu erstellen. Arbeitet man zusätzlich mit Fragments können komplexere User Interfaces gebaut werden, welche modular in einzelne Bereiche aufgeteilt werden. Dies ist vor allem hilfreich um für unterschiedliche Auflösungen ein ansprechendes Layout zu präsentieren.

Arbeitet man mit MvvmCross, ist es zwingend diese Activities im Namensraum „Views“ liegen, da die ViewModels die zugehörigen Views explicit in diesem Namensraum suchen.

## Android User Interface Design

Wie bereits mehrfach erwähnt, erhält man mit Xamarin keine Umgebung, welche einem die Entwicklung des User Interface erleichtert. Grundsätzlich ist es sehr wichtig, dass man die Architektur und die Eigenheiten der Android Entwicklung beherrscht um ein gutes Benutzererlebnis zu erreichen.

### Databinding mit MvvmCross in Android

Die ViewModels können unterschiedlich an die Views gebunden werden, wovon zwei Möglichkeiten folgend beschrieben werden.

**Swiss**

Die Swiss Binding Syntax ist ähnlich, wie man es vom Windows Phone her kennt.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:local="http://schemas.android.com/apk/res-auto"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <Mvx.MvxListView
        android:id="@+id/mvx.MvxListView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        local:MvxBind="ItemsSource SliceInfoCollection; ItemClick ListItemClickedCommand"
        local:MvxItemTemplate="@layout/item_slice"
        android:layout_gravity="center_vertical" />
</LinearLayout>
```

Im oben gezeigtem XML-Layout File wird eine einfach ListView dargestellt. Die Properties des dazugehörigen ViewModels können nun über local:MvxBind=““ an die View gebunden werden.

Das Prinzip ist immer das gleiche: $Target$ $SourcePath$

$Target$ zeigt auf das Property auf der View (ItemSource)
$SourcePath$ zeigt auf das Property auf dem ViewModel (SliceInfoCollection)

Wird zusätzlich ein ValueConverter benötigt, kann dieser hinten angehängt werden.

Beispiel: local:MvxBind=“Text Name, Converter=StringConverter“

Werden mehrere Bindings auf einer Komponente benötigt, können mittels Semikolon die weiteren Bindungen hinzugefügt werden. Im oben gezeigten Beispiel wird der ItemClick Event mit dem ListItemClickCommand gebunden. Dadurch erreicht man ein schlankes User Interface und eine klare Trennung der Schichten.

**Fluent**

Mit dem Fluent Syntax kann die Datenbindung mit Hilfe von Lamda-Expressions gemacht werden, wie man sich dies in C# gewohnt ist. Dies ist vor allem dann von nöten, wenn man kein Layout File hat.

Beispiel:

```
var set = this.CreateBindingSet<MyView, MyViewModel>();
set.Bind(nameLabel).For(v => v.Text).To(vm => vm.Customer.FirstName);
```

Oft wird der Fluent Syntax bevorzugt, da bereits beim Kompilieren fehlerhafte Verweise erkennt werden. Bei der Swiss Syntax, wo die Zuweisung innerhalb des XML passiert, kann der Compiler dies nicht.

## Persönliche Erfahrungen mit Android

Am Anfang bereitete mir die Installation von Xamarin viel Mühe. Da ich Cross Plattform Applikationen entwickeln wollte, installierte ich alle nötigen Komponenten auf meinem Computer. Nach der Installation des Android SDK verhielt sich mein Computer bei der Windows Phone Entwicklung nicht mehr wie gewünscht. Beim Kompilieren und Starten des Windows Phone Emulator endete ich jeweils in einem Bluescreen of Death. Etliche Stunden suchte ich im Internet eine Lösung, wie ich beide Emulatoren auf der gleichen Maschine zum Laufen bringen konnte. Leider konnte keine befriedigende Lösung gefunden werden.
An einem Vortrag über Xamarin von Thomas Kälin, Software Entwickler bei der bbv Solutions AG, wurde diese Erfahrung geteilt. Es wurde mir empfohlen für die Windows Phone Entwicklung eine VM einzusetzen und nur auf dieser Hyper-V zu aktivieren. Scheinbar verträgt sich Hyper-V parallel mit VMWare nicht.

Somit hatte ich also für die Cross Plattform Entwicklung zwei VM am Laufen. Zum einen die notwendige OSX VM für die iOS Entwicklung und nun auch noch eine VM mit Windows 8.1 für die Windows Phone Entwicklung.

Die Android Entwicklung hat sich nach der ganzen Konfiguration als sehr erfreulich erwiesen. Nachdem diese ersten Hürden genommen wurden, konnte relativ schnell gute Ergebnisse erzielt werden.

# iOS Applikation

## Einführung

Die Entwicklung mit OS X war für mich komplett neu. Jedoch war ich sehr neugierig und freute mich darauf einen Prototypen zu erstellen. Für die Umsetzung hatte ich nicht viel Zeit eingeplant und beschränkte mich daher darauf, eine einfache lauffähige Version hinzukriegen.
Zudem erhoffte ich mir, von den Erfahrungen der Android Entwicklung profitieren zu können.

## Infrastruktur

Da Xamarin.iOS nicht auf einem Windows Rechner kompiliert werden kann, ist eine Virtuelle Maschine mit OS X oder ein Apple Rechner zwingend. Anfangs wurde mit einem Mac Mini gearbeitet, aber später aus Mobilitätsgründen doch auf eine VM gewechselt.

Folgende Installationen sind auf der VM nötig:

- OS X (Mindestens OS X Lion 10.7)
- XCode Entwicklungstools
- Xamarin Studio

Mit der Installation von Xamarin wurde ein Build Host installiert. Wenn dieser gestartet wird, kann von einem Computer im gleichen Netzwerk eine Verbindung hergestellt werden und es findet ein sogenanntes „Pairing“ statt.

Leider ist es nicht möglich, mehrere Verbindungen mit dem gleichen Server zu haben. Es kann also immer nur ein Projekt verbunden werden. Da man oft auch Beispiele kompilieren will, benötigt das wiederholte Pairing enorm viel Zeit.

## Persönliche Erfahrungen mit iOS

Die iOS Entwicklung war sehr herausfordernd. Aufgrund des gesteckten Zeitrahmens konnte ich mich mit der Architektur und dem Lebenszyklus von iOS Applikationen nicht wie gewünscht auseinandersetzen. Der Prototyp konnte dank guter Code Beispiele umgesetzt werden. Im Nachhinein wäre es wohl klüger gewesen, sich von Anfang an für eine Plattform zu entscheiden und nicht beide Plattformen anzuschauen.

Xamarin erlaubt die iOS Entwicklung mit C#. Versteht man die darunterliegende Architektur von iOS nicht, bringt einem das jedoch nicht sonderlich viel. Bei der Android Version konnten Java Implementationen dank gutem Verständnis adaptiert werden. Nun stand mit Objectiv-C und der XCode Entwicklungsumgebung eine unbekannte Komponente im Weg, welche zuerst erlernt werden musste.

Für weitere Projekte würde ich empfehlen, sich zuerst mit einer komplett nativen iPhone Applikation auseinanderzusetzen. Kennt man nämlich die Eigenheiten der iOS Entwicklung, ist die Umsetzung mit Xamarin und C# viel einfacher. Dies gilt natürlich auch für die Android Entwicklung.

## Aufgetretene Probleme bei der iOS Entwicklung

### Error beim Starten der Applikation

Beim Starten der Applikation kam immer der Fehler, dass die Versionen auf dem Build Server und dem Client von Xamarin nicht gleich sind. Dies tauchte jedoch nicht deswegen auf, sondern weil auf der VM eine andere Uhrzeit eingestellt war als lokal.

Nachdem die beiden Zeiten „synchronisiert“ wurden, verlief der Build Vorgang problemlos.

### Verbindungs Probleme with Build Host

Ab und zu tauchen Verbindungsprobleme mit dem Build Host auf. Oder der Build Host konnte nicht erreicht werden. Gemäss weiteren Erfahrungen von Thomas Kälin (bbV Solutions) ist dies auch bei Ihnen ein bekanntes Problem. Oft half jedoch ein Neustart des Build Hosts.

### Probleme mit NuGet im Xamarin Studio

NuGet ist nicht im Xamarin Studio integriert, kann aber über ein AddIn auch auf OS X installiert werden. Nach der Installation trat folgender Fehler auf:

This project refrences NuGet package(s) that are missing on this computer. Enable NuGet Package Restore to download them.

Das Problem liegt am Microsoft.Bcl Package. Daher dass ich asynchrone Aufrufe mit Hilfe des Microsoft.Net.Http Packages mache, kommt automatisch das Microsoft.Bcl.Build Assembly über NuGet in das Projekt. Beim Kompilieren probiert dieses Assembly Konfigurations Files zu schreiben. Dies wurde Mono nicht hinzugefügt und funktioniert daher nur  auf einem Windows Rechner.

Um nicht länger Zeit zu verschwenden, starte ich die OS X Applikation nicht direkt aus dem Xamarin Studio auf der VM, sondern auch vom Visual Studio über den Build Host.

### Failed to Clean Build Cache

Beim wiederholten Debuggen der Applikation kann ab und zu der Build Cache nicht gelöscht werden. Nach einem Restart der Virtuellen Maschine funktionierte dies wieder.

## Testing

Als Testing Framework wurde NUnit gewählt. Zum Simulieren von gewissen Services wurde zudem das Mocking Framework Moq verwendet.

Da der IoC Container von MvvmCross in den ViewModels verwendet wird, musste auch dieser in den Tests verfügbar sein. Um dies zu erreichen stellt MvvmCross ein Assembly zur Verfügung.

Alle Test Klassen haben in der Vererbungshierarchie MvxIoCSupportingTest implementiert. Dadurch steht das Property „Ioc“ zur Verfügung und kann zum Registrieren von Klassen verwendet werden.

Die Methode CreateMockNavigation instanziert einen eigenen MockDispatcher und registriert diesen im Ioc Container. Dadurch kann nun auch die Navigation simuliert werden.

```
public class TestBase : MvxIoCSupportingTest
{
    protected MockDispatcher CreateMockNavigation()
    {
        var dispatcher = new MockDispatcher();
        Ioc.RegisterSingleton<IMvxMainThreadDispatcher>(dispatcher);
        Ioc.RegisterSingleton<IMvxViewDispatcher>(dispatcher);
        return dispatcher;
    }
```

Beispiel Test:

```
[Test]
public void TestLocationCommand()
{
    base.ClearAll();

    var mockdispatcher = CreateMockNavigation();
    CreateMockForApplicationServices();
    var mockCalculationService = MockCalculationService();
    var model = new RegionPlaceViewModel(mockCalculationService.Object);

    //Action
    model.LocationCommand.Execute(null);

    //Assert
    Assert.AreEqual(1, mockdispatcher.Requests.Count);
    var request = mockdispatcher.Requests[0];
    Assert.AreEqual(typeof(LocationViewModel), request.ViewModelType);
}
```

Dieser Test prüft, ob der LocationCommand das richtige ViewModel aufruft.

## Persönlicher Bericht

Die mobile Entwicklung hat mich schon länger interessiert und ich begann voller Enthusiasmus diese Studienarbeit. Vor dieser Studienarbeit hatte ich erst einmal eine native Applikation für Android entwickelt und war jetzt interessiert, wie Xamarin mit C# als Programmiersprache plattformübergreifend genutzt werden kann. Die .NET Umgebung war mir aus beruflichen Gründen bereits bekannt. Ich war sehr erstaunt wie viele Beispiele und gute Dokumentationen auf der Webseite von Xamarin zu finden sind, merkte jedoch auch schnell, dass dies absolut nötig ist um Fuss zu fassen in der Cross Plattform Entwicklung.

Unerwartet viel Aufwand bedeutete die Konfiguration der Entwicklungsumgebung, da für die iOS Entwicklung ein OS X Rechner oder eine virtuelle Maschine benötigt wird.
Auch Hyper-V für die Windows Phone Entwicklung zu aktiveren wollte nicht auf Anhieb funktionieren und endete beim Starten des Emulators in einem Bluescreen of Death (BSOD). Dies führte dazu, dass auch die Windows Phone Applikation auf einer VM getestet wurde, da Hyper-V parallel mit VMWare zu Problemen führte.

Bezüglich der Umsetzung seitens der MIT Innovation AG wurde mir relativ viel Freiraum gelassen. Gewünscht war die Portierung auf Android und iOS und auch ein Redesign der aktuellen Windows Phone Applikation. Ich wusste, dass die Entwicklung eines Prototyps für Android und iOS viel Zeit benötigt und habe mich daher nur für die Portierung entschieden, auch mit dem Hintergedanken, dass ein Erfahrungsbericht auf diesen Plattformen für die MIT Innovation AG nützlicher ist.

Im Nachhinein wäre es wohl klüger gewesen, ich hätte mich nur mit der Android Entwicklung befasst. Je länger das Projekt voranschritt und ich bestimmte Funktionalitäten in der Android Applikation integrieren wollte, bemerkte ich meine fehlenden Kenntnisse. Diese zu vertiefen lies aber der zeitliche Rahmen nicht zu, da ich den iOS Prototypen noch entwickeln wollte. In dieser Phase bemerkte ich, wie wichtig ein Austausch mit einer weiteren Person wäre. Zum einen kann man die Arbeit gezielt aufteilen und zum andern können durch den fachlichen Austausch viele Fehler im Voraus verhindert werden.

Etwa in der Hälfte meiner Arbeit begann die MIT Innovation AG mit einer weiteren Portierung auf Android. Es war erfreulich zu hören, dass meine bisher gewonnenen Kenntnisse bereits einfliessen konnten. Dies war für mich ein Glücksfall, da ich mich nun mit einem anderen Entwickler austauschen konnte.

Xamarin als Entwicklungsumgebung hat mich sehr beeindruckt und dennoch einen faden Nachgeschmack hinterlassen. Dank portablen Klassenbibliotheken ist es möglich, viel Code für alle Plattformen gemeinsam zu nutzen, jedoch müssen die User Interfaces pro Plattform separat programmiert werden. Dies war sehr aufwändig und benötigte gute Kenntnisse der Architektur und dem Lebenszyklus von Android, iOS und dem Windows Phone.

Ich werde sicherlich Xamarin beruflich weiterverfolgen. Diese Arbeit hat mir jedoch gezeigt, dass es sinnvoll wäre zuerst eine native Applikation mit Java oder Objectiv-C zu programmieren, um die Vorteile von Xamarin besser abschätzen zu können.

# Literaturverzeichnis

[1] Gartner (February 2014)
http://www.gartner.com/newsroom/id/2665715

[2] Xamarin Cross Plattform Application
http://docs.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/case_study-mwc/

[3] Portable Class Abstraction
http://msdn.microsoft.com/de-de/library/gg597391(v=vs.110).aspx

[4] Dependency Injection, Inversion of Control
http://joelabrahamsson.com/inversion-of-control-an-introduction-with-examples-in-net/

[5] Xamarin, Sharing Code Options
http://docs.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/sharing_code_options/

[6] MSDN, MVVM Pattern, Portable Class Library
http://msdn.microsoft.com/de-de/library/hh563947(v=vs.110).aspx

[7] GitHub, MvvmCross PlugIn
https://github.com/MvvmCross/MvvmCross/wiki/MvvmCross-plugins#writing-a-plugin