Visual Studio

# Das Visual Studio Inselbuch

## Die Sommerlektüre für Entwickler

Microsoft Press

# Inhaltsverzeichnis

# Vorwort

Liebe Visual Studio-Freunde und -Freundinnen,
Sie kennen das: Es ist Urlaubszeit - man liegt am Strand, oder die Familie besucht das x-te Museum - und so langsam kehrt die Langeweile ein. In dieser Situation schafft das „Visual Studio Inselbuch" Abhilfe.

Wir haben führenden Experten aus der Microsoft-Entwickler-Community die klassische Frage gestellt: „Welche aktuelle Technologie würdest Du auf eine einsame Insel mitnehmen?". Dann haben wir alle Einsender gebeten, einen aktuellen Artikel dazu auszuwählen. Das Resultat ist eine bunte Mischung interessanter Informationen, die sich bestens zum Wissen-Update im Liegestuhl eignet. Ideal, um eventuell im Projektstress entstandene Wissenslücken entspannt aufzufüllen.

Sie wissen noch nicht, wohin Sie das Inselbuch mitnehmen sollen? Auch dafür haben wir Abhilfe – unsere Autoren haben als Inspiration auch noch Urlaubstipps parat. Somit wäre die Packliste für die Reise komplett:

1. Handtuch (optional)
2. eBook-Reader
3. Visual Studio Inselbuch

Bleibt uns nur noch, uns bei allen Mitwirkenden für den großen Einsatz so kurz vor der Sommerpause zu bedanken. Das Visual Studio Team der Microsoft Deutschland wünscht Ihnen erholsame Urlaubstage!

Herzlichst

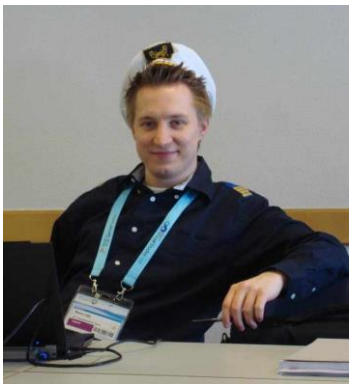
Nina Steiner und Uwe Baumann
Produktmanagement Developer Tools
Microsoft Deutschland GmbH

# Kapitel 1: Team Foundation Server TFS

## Neno Loje & Thomas Schissler – Trends in der modernen Software-Entwicklung

### Name

• Neno Loje



### Arbeitgeber

• Freiberuflich tätiger Berater & MVP für Visual Studio ALM

### Technologieschwerpunkte

• Team Foundation Server (TFS) – zentrale Verwaltung von Projekten

• Application Lifecycle Management (ALM) – von der Idee bis zum Release

• Moderne Softwareentwicklungsmethoden (Scrum) – wie kann man mehr erreichen?

### Mein Urlaubstipp

Natürlich "Rosario" auf der Insel "Orcas".

### Meine Inseltechnologie

Die Erkenntnis, dass keine Technologie oder kein Werkzeug Kommunikation zwischen Menschen ersetzen kann und dann das man sich im gleichen Team, indem man sich anders organisiert, deutlich mehr erreichen kann.

### Meine Sommerlektüre

"Build-Measure-Learn" oder: Trends in der modernen Software-Entwicklung

## Name

• Thomas Schissler

## Arbeitgeber

• artiso AG

## Technologieschwerpunkte

• Steigerung der Zufriedenheit bei Entwicklern und Kunden durch agile Software-Entwicklung
• Einsatz des Team Foundation Servers um moderne Software-Entwicklung zu ermöglichen
• Coaching von Teams, POs und Stakeholder um ihr Mindset in Richtung Agilität zu erweitern
• Vermittlung verschiedener Test-Praktiken und Architektur-Patterns, damit der Spass am Software-Projekt nachhaltig ist

## Mein Urlaubstipp

Nicht so weit weg fahren. Der Stress lohnt sich nicht und entspannen kann man überall.

## Meine Inseltechnologie

Meine Inseltechnologie ist ein Strick und zwar der an dem ein Team gemeinsam zieht um gemeinsam Spaß zu haben und grandiose Ergebnisse zu erzielen ohne sich von irgendjemand oder irgendetwas dabei aufhalten zu lassen.

## Meine Sommerlektüre

"Build-Measure-Learn" oder: Trends in der modernen Software-Entwicklung

## Trends in der modernen Software-Entwicklung

*Wie Software-Entwicklung mit Hilfe moderner Methoden schneller, effizienter und zielgerichteter organisiert werden kann*

## Build-Measure-Learn

Software-Entwicklung ist seit jeher ein Gebiet mit rasanten Veränderungen und Entwicklungen. Doch während sich in der Vergangenheit diese Entwicklungen zumeist auf einer technologischen Ebene abgespielt haben, erleben wir momentan die größten Veränderungen im Bereich der Prozesse und der Organisation der Softwareentwicklung. Vor allem finden agile Methoden immer stärker Einzug in Projekte aller Art, auch wenn die Interpretation von Agilität noch sehr unterschiedlich ist.

Dabei spielt ein Konzept eine besonders wichtige Rolle, welches als Build-Measure-Learn bezeichnet wird. Ziel ist dabei, durch kurze Zyklen nach Bewertung eines Zwischenergebnisses Entscheidungen für die nächsten Entwicklungsschritte zu treffen und so den Nutzen der Entwicklung zu optimieren. So einleuchtend sich dieses Konzept anhört, so herausfordernd ist jedoch die richtige Umsetzung. Viele Bereiche der Software-Entwicklung sind direkt oder indirekt betroffen und um dieses Konzept tatsächlich erfolgreich anwenden zu können, sind etliche tiefgreifende Veränderungen notwendig.

Wir wollen uns zunächst das Build-Measure-Learn-Konzept etwas näher anschauen und betrachten dann in den weiteren Kapiteln dieses Artikels die wichtigsten Veränderungen, die zur Umsetzung notwendig sind.
Während in der Vergangenheit versucht wurde, ein Software-Projekt möglichst gut zu planen und diesen Plan dann möglichst gut umzusetzen, geht Build-Meaure-Learn davon aus, dass wir erst im Laufe der Zeit bestimmte Fragen richtig beantworten können. Statt also einen Plan zu erstellen und diesem zu folgen, wird entschieden, was aus der aktuellen Situation heraus der nächste geeignete Schritt ist, der den größten Nutzen generiert. Dabei spielt der Kunde eine wesentlich größere Rolle, da er durch sein Feedback direkten Einfluss auf diese Entscheidungen hat.

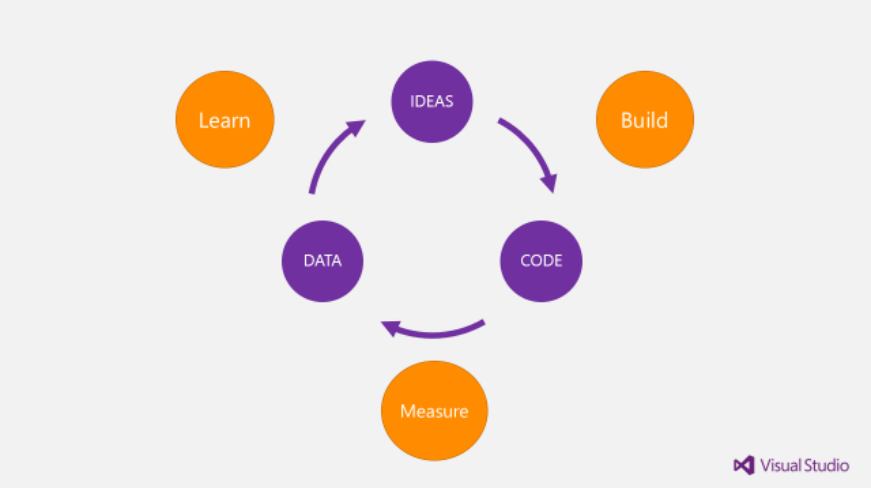



*Abbildung: Der Build-Measure-Learn Ansatz beschreibt, wie in kurzen Zyklen Ideen realisiert werden, um aus den Erfahrungen heraus die weitere Produktentwicklung zu planen.*

Das Konzept von Build-Measure-Learn sieht dabei vor, dass eine Idee zunächst als Funktionalität in der Anwendung umgesetzt wird (Build). Diese Funktionalität wird den Anwendern oder einer kleinen Test-Gruppe zur Verfügung gestellt und wir ermitteln über den Schritt Measure Daten bezüglich des Nutzens dieser Funktion. Anschließend erfolgt der Schritt Learn, bei dem wir aus diesen Ergebnissen lernen, wie wir unsere Software-Anwendung verbessern können. Diese Verbesserungs-Ideen werden wieder umgesetzt, gemessen, daraus eine Lehre gezogen und so weiter.

Als Beispiel könnte der Betreiber eines Internet-Portals für Flugbuchungen eine Funktion einbauen, die es bestimmten Benutzern erlaubt, direkt nach der Buchung des Fluges ein Hotel am Zielflughafen zu reservieren. Anschließend würde gemessen werden, ob Anwender dieser Funktion eine Hotelreservierung häufiger vornehmen, als Anwender die diese Funktion separat aufrufen müssen. Ausgehend von diesen Erkenntnissen kann die Funktion weiter ausgebaut oder auch wieder eingestellt werden.

Damit wird auch akzeptiert, dass nicht alle Annahmen in der Software-Entwicklung korrekt sind und dass wir auch Fehler machen. Build-Measure-Learn ist dann effizient, wenn man es in kurzen Zyklen anwendet. Ein Beispiel dafür findet man z.B. bei Microsoft, wo die aktuelle Version des TF Service alle 3 Wochen aktualisiert wird und das Kundenfeedback direkten Einfluss auf die Weiterentwicklung einzelner Funktionen hat.

Damit bietet das Learn die Möglichkeit, die Funktionalität unserer Anwendung auf das zu fokussieren, was den Anwendern tatsächlichen bzw. den meisten Nutzen bringt und ggf. eine Funktionalität, die nicht genutzt wird, wieder zu entfernen. Um das Measure auszuführen, stehen verschiedene Methoden zur Verfügung. So kann z.B. das Anwenderfeedback direkt ausgewertet werden, wie es Microsoft mit dem UserVoice-Portal für Visual Studio (http://visualstudio.uservoice.com) vormacht. Oder durch Applikationsmetriken kann direkt gemessen werden, wie häufig eine bestimmte Funktion genutzt wird. Mit diesen Informationen erhält man eine gute Basis, Entscheidungen im Sinne des maximalen Kundennutzens zu treffen. Diese Aspekte stellen natürlich besondere Anforderungen an die Build-Phase, um in diesen kurzen Zyklen qualitativ hochwertigen und wartbaren Code zu liefern.

## Fazit

Der Anspruch, die perfekte Software zu planen und anschließend umzusetzen, muss heute als unrealistisch angesehen werden, egal wie viel Planungsaufwand investiert wird. Als viel erfolgreicher hat sich herausgestellt, durch kleine Experimente Funktionalität in sehr kurzen Zyklen hinzuzufügen und dann herauszufinden, wie die Software weiter optimiert werden kann. Also weg vom Planen, hin

zum Ausprobieren und sich eingestehen, dass die eine oder andere Idee vielleicht doch nicht so gut war.

Siehe auch Moderne Softwareentwicklung 1 - Trends in der modernen Softwareentwicklung

## Modernes Anforderungsmanagement und Planung

Bevor das Entwicklungs-Team mit der Arbeit beginnen kann, werden die Wünsche und Ideen der einzelnen Stakeholder (Anspruchsgruppen wie Auftraggeber, Anwender, usw.) gesammelt und in eine Form gebracht, die die spätere Umsetzung begünstigt und das Ergebnis möglichst klar beschreibt.

### Product Ownership

Die Koordination der Ideensammlung und die Formulierung einer Vision und der gewünschten Ergebnisse werden durch einen zentralen Produktverantwortlichen, den so genannten Product Owner (PO) unterstützt. Dieser arbeitet auf der einen Seite mit den Stakeholdern, um die Ideen zu erfassen, und zum anderen dient er dem Entwicklungs-Team als zentraler Ansprechpartner bei fachlichen Fragen. Ebenso kommuniziert der PO die Vision, also für wen die Software entwickelt wird (Zielgruppe) und welcher Nutzen mit ihr realisiert werden soll (Business Value).
Dabei geht es nicht um viele Worte, sondern im Gegenteil eher um nur so viel Text, wie nötig mit dem „richtigen" Detailgrad, um das gewünschte Resultat klar zu kommunizieren. Eine beliebte Form, die sich in den letzten Jahren etabliert hat, ist Formulierung von Anforderungen in Form einer „User Story".

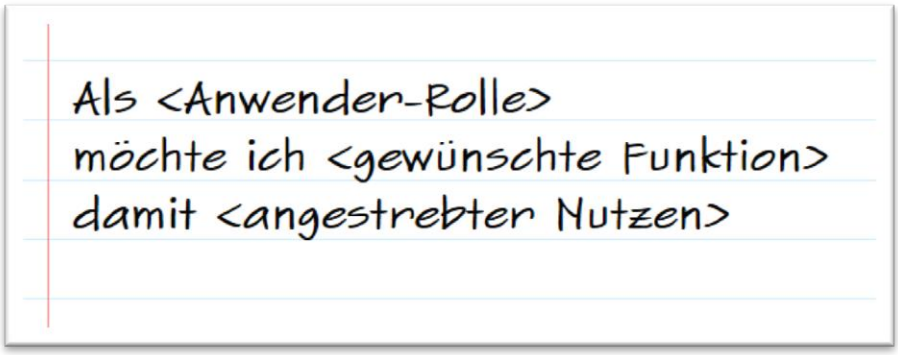



*Abbildung: Die User Story-Syntax beschreibt in einer kompakten Weise das erwartete Ergebnis aus Sicht des Benutzers sowie welcher Nutzen für den Benutzer realisiert werden soll.*

### Product Backlog

Alle gesammelten Ideen kommen – häufig noch ungefiltert – auf eine große Liste, das so genannte Product Backlog. Da es bei den meisten Projekten mehr Wünsche gibt, als das Team in der Lage wäre direkt umzusetzen, wird die Liste zu einer Art Warteschlange. Die Anforderungen in der Warteschlange

in der Reihenfolge, wie sie eingetragen wurden, einfach abzuarbeiten ist dabei allerdings häufig nicht die effizienteste Wahl. Aus diesem Grunde bewertet der Produktverantwortliche in Zusammenarbeit mit den Stakeholdern und dem Entwicklungs-Team das Product Backlog und sortiert es neu, sodass immer zuerst das „Wichtigste" abgearbeitet wird.

Dazu braucht es vor allem zwei Informationen zu jedem Element: die Größe und den Nutzen für den Benutzer bzw. das Unternehmen. Die Größe wird vom Entwicklungs-Team in Form einer relativen Schätzung beigesteuert, d.h. es werden Elemente auf dem Product Backlog verglichen, ohne konkret über die Umsetzung nachzudenken (Beispiel: „Ich weiß zwar nicht, wie viel ein Apfel wiegt, aber wenn ich ihn mit einer Erdbeere vergleiche, dann ist es mindestens Faktor 5."). Für den Mehrwert/Nutzen fragt man möglichst die Personen, die es am besten wissen sollten: die Stakeholder. Auch hier kann man die Elemente relativ gegeneinander abwägen (Feature A ist aus Kundensicht drei Mal so viel wert wie Feature B). Weitere Kriterien könnten noch das Risiko (besonders riskante Anforderungen möchte man vielleicht möglichst frühzeitig abarbeiten) bzw. Abhängigkeiten zwischen den Elementen sein.

## Entwicklungs-Team

Die Umsetzung der Elemente vom Product Backlog (Product Backlog Items) erfolgt durch das Entwicklungs-Team. Dieses ist idealerweise nicht zu groß, sodass man sich gut untereinander abstimmen kann, am gleichen Ort angesiedelt und verfüget über alle Fähigkeiten, die benötigt werden um die Software fertigzustellen, inklusive aller Qualitätsaspekte und Testing. Die Anzahl der Product Backlog Items, die das Team innerhalb einer Zeiteinheit von z.B. einer Woche fertigstellt, ist dann die Geschwindigkeit des Teams (Velocity). Mit dieser Zahl lassen sich Prognosen anstellen, die auf der bisher gemessenen Geschwindigkeit basieren. So kann der Product Owner bereits frühzeitig kommunizieren, welche Features man bis wann erwarten kann und die Erwartungen richtig setzen.

Einige Teams setzen sich ein „Work in Progress"-Limit und definieren somit, dass z.B. nur fünf Elemente gleichzeitig vom Team bearbeitet werden dürfen. Die Anzahl wird häufig so gewählt, dass es ausreicht das Team kontinuierlich zu beschäftigen, aber eben auch nicht zu viel, sodass das Team einen klaren Fokus behalten kann.
Dabei ist das Team dafür verantwortlich, den Code nachhaltig zu schreiben und sich keine „technische Schuld" aufzuladen. Ziel sollte sein, dass man auch sechs Monate später noch gerne in dem Code arbeitet. Leidet die Codequalität bzw. wurde nicht darauf geachtet erweiterbaren und testbaren Code zu schreiben, wird es immer aufwendiger Änderungen durchzuführen. Die Umsetzung neuer Funktionen dauert dann zunehmend länger und bremst mittelfristig die Geschwindigkeit des Team aus (bis hin zu praktisch Null).

## Fazit

Ein für alle sichtbares Product Backlog dient als Spiegel dessen, was das Team vor hat bzw. woran gearbeitet wird. Voraussetzung ist, dass auch wirklich alle Punkte auf dem Product Backlog notiert werden. Es bietet die ideale Grundlage für Diskussionen, da die Reihenfolge für alle Beteiligten transparent erkennbar ist. Es sind aber vor allem auch die Diskussionen, die stattfinden, während Elemente für das Product Backlog geschrieben oder verfeinert werden, die letztlich zu einem Erkenntnisgewinn der Beteiligten führen.

Jede Rolle hat einen klaren Fokus:

- Der Product Owner als zentraler Ansprechpartner und Verantwortlicher für die Sortierung des Product Backlogs repräsentiert die Stakeholder. Er möchte möglichst viele Features haben (und ist dafür verantwortlich mit dem Kunden zu erarbeiten, welche das sind).
- Die Stakeholder sind Personen, die an dem Projekt ein wesentliches Interesse haben und in irgendeiner Weise Einfluss darauf nehmen möchten, z.B. Anwender, Management, Marketing, Support etc.
- Das Entwickler-Team ist dafür verantwortlich, die Anforderungen richtig umzusetzen. Dazu zählt auch die Verantwortung für die Nachhaltigkeit, also für eine wartbare, erweiterbare Codebasis.
- Einige Teams haben noch einen Prozessbegleiter (bei Scrum als "Scrum Master" bekannt), der das Team coacht, dabei hilft die Zyklen – von der Idee zum fertigen Feature – zu verkürzen bzw. die Geschwindigkeit und damit den Durchsatz des Teams weiter zu erhöhen.

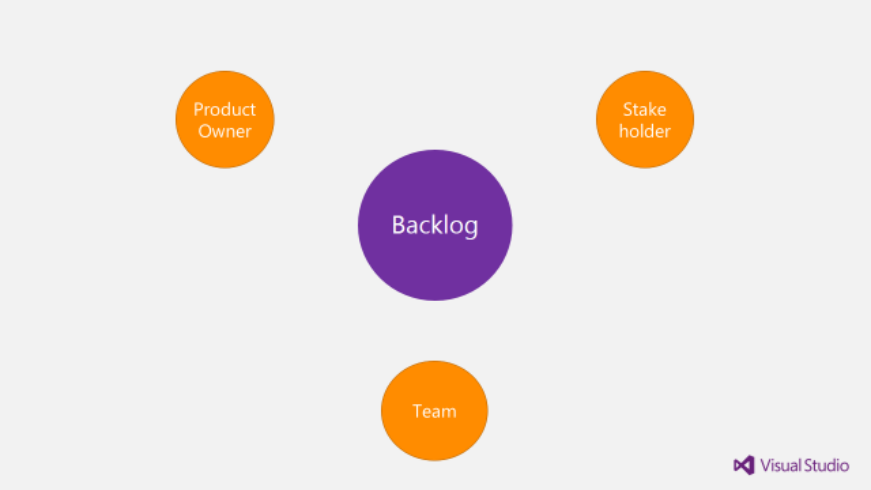



*Abbildung: Durch das Backlog gewinnt das Team an Transparenz (jeder sieht, was gerade getan wird). Durch die klaren Zuständigkeiten kann sich jede beteiligte Person voll und ganz auf ihre Aufgabe konzentrieren.*

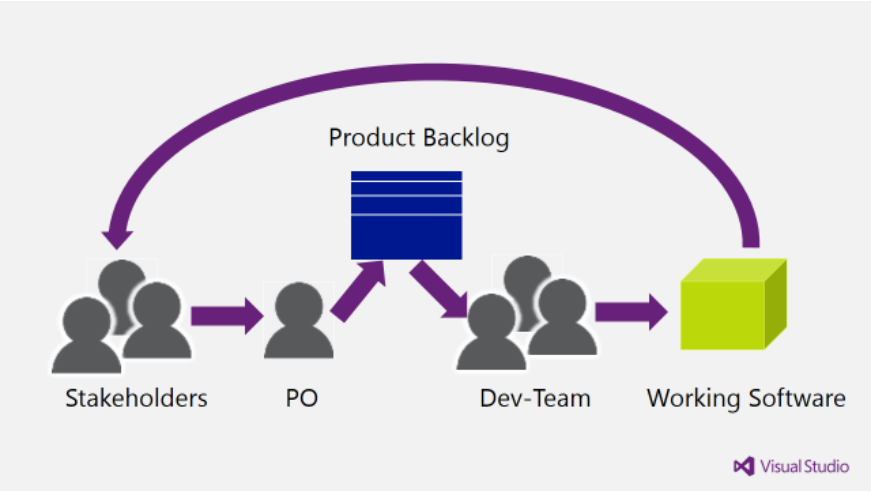
Siehe auch Moderne Softwareentwicklung 2 - Modernes Anforderungsmanagement und Planung

## Stakeholder effizient einbinden

Ein sehr wichtiger Aspekt moderner Software-Entwicklung ist die Einbeziehung von Kundenfeedback in den Entwicklungsprozess. Wobei der Begriff „Kunde" dabei etwas weiter zu fassen ist. Im Englischen wird dafür der Begriff „Stakeholder" benutzt, womit alle Personen bezeichnet werden, die an einem Projekt ein Interesse haben, also neben den klassischen Anwendern eben auch Management, Marketing, Vertrieb etc.

### Stakeholder Feedback nutzen

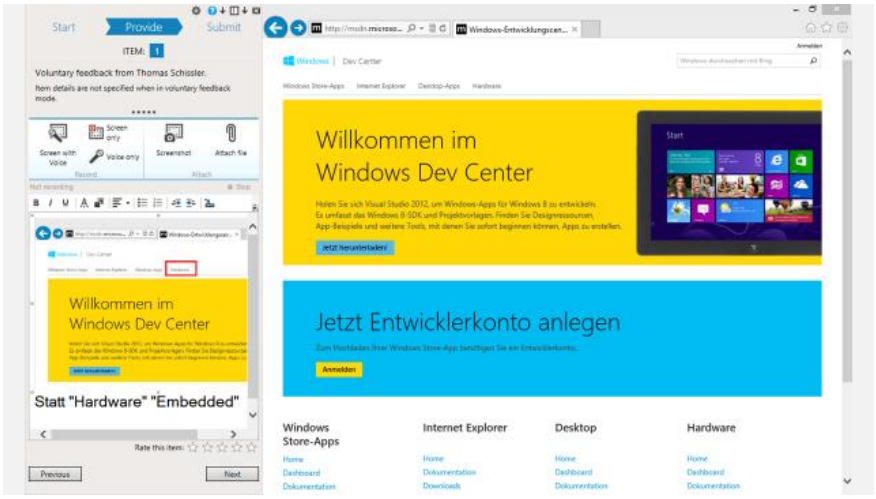
Die Stakeholder kommunizieren ihre Wünsche an den Product Owner (PO), der diese in einem Backlog sammelt, verwaltet und auf Basis des Inputs seiner Stakeholder entscheidet, welche Funktionen als nächstes implementiert werden sollen. Diese Funktionen werden im Entwicklungs-Team umgesetzt und am Ende einer kurzen Iteration als nutzbares Software-Inkrement bereitgestellt. In der Praxis fehlt häufig der nun folgende - entscheidende - Schritt, nämlich dass sich die Stakeholder und der PO mit dem Ergebnis befassen und aufgrund der daraus gewonnenen Erkenntnisse ihre Anforderungen anpassen. Die Stakeholder sollen also nicht - wie in einer Einbahnstraße - ihre Anforderungen nur in diesen Prozess einbringen, sondern sich auch mit dem Ergebnis beschäftigen um zu sehen, wie die Software aktuell aussieht und daraus neue Erkenntnisse zu ziehen.




Aber den Stakeholdern wird es oftmals nicht ganz einfach gemacht, diese Feedback-Schleife zu schließen. Es ist meist nicht ihre primäre Aufgabe, sich um ein Software-Projekt zu kümmern. Sie sind keine Software-Experten und die Entwicklungsteams sehen ihre Aufgabe oft als erledigt an, wenn sie sagen können: „Wir haben die aktuellste Version auf einem Netzlaufwerk abgelegt, man kann sich diese von dort herunterladen und installieren." Mit den Problemen, die dabei entstehen können, wollen Stakeholder sich nicht herumschlagen.

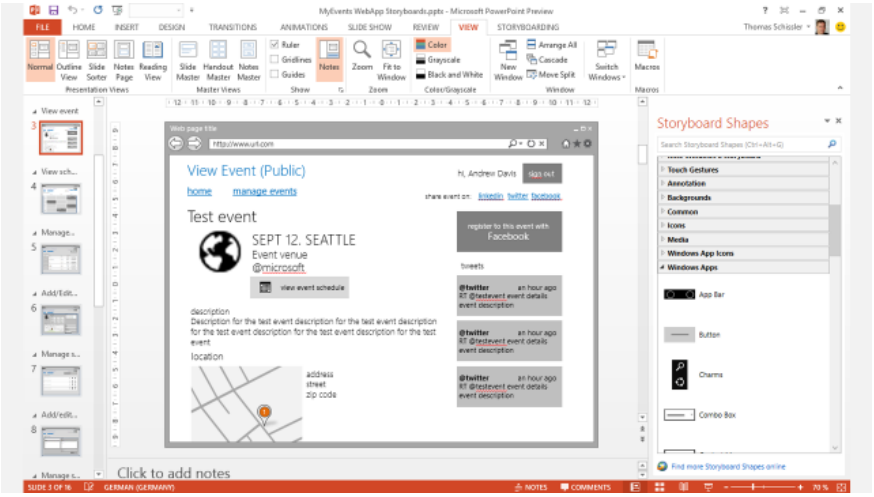
Wichtig ist hierbei, dass Entwicklungsteams die Verantwortung anerkennen, dass sie ihre Stakeholder beim Liefern von Feedback bestmöglich unterstützen müssen. Entwickler haben die Kompetenz und die Werkzeuge, die dabei helfen können und - noch viel wichtiger - die Entwickler sind auf dieses Feedback angewiesen. Stakeholder sollten also nicht als „Störfaktor" gelten, die eh nicht genau wissen, was sie wollen, sondern sie sind ja die Daseinsberechtigung für Software-Entwickler. Denn nur wenn die Stakeholder einen Nutzen durch die Software haben, wird auch Geld für die Entwicklung bereitgestellt. Es liegt also in der Verantwortung der Software-Entwickler, die Kommunikation mit den Stakeholdern zu optimieren und so für optimale Ergebnisse zu sorgen. Durch geeignete Werkzeuge können wir diesen Prozess noch weiter unterstützen.

So bietet z.B. der Team Foundation Server zusammen mit dem Microsoft Feedback Client die Möglichkeit, dass Stakeholder dieses Feedback direkt über eine kleine Anwendung geben können, entweder über Texte mit Screenshots oder eine Video-Aufzeichnung mit Audio. Dieses Feedback kann vom Entwicklungs-Team am Ende einer Iteration eingefordert und verwaltet werden. Es ist beispielsweise jederzeit nachvollziehbar, wie ein entsprechendes Feedback sich in Änderungen der Anforderungen ausgewirkt hat oder welcher Stakeholder noch gar kein Feedback gegeben hat.




## Storyboards

Ein weiterer Aspekt für die Kommunikation zwischen dem Entwicklungs-Team und den Stakeholdern ist die Frage, wie Anforderungen und Wünsche von Anwendern effizient formuliert werden können. Neben einer schlanken textlichen Beschreibung z.B. in Form von User Stories inkl. Akzeptanz¬Kriterien nutzen immer mehr Teams auch Storyboards. Diese stellen in vereinfachter und bildhafter Form dar, wie ein Anwender mit der zu implementierenden Funktionalität arbeiten wird, ähnlich zu einem Storyboard zu einem Film. Das Ziel ist dabei, mit einigen einfachen Bildern ein einfaches Verständnis für den Ablauf zu geben.




Diese Form der Dokumentation können Stakeholder und POs nutzen, um ihre Anforderungen klarer zu beschreiben und damit die Chance zu erhöhen, gleich bei der ersten Implementierung ihre Vorstellungen zu treffen. Oder Teams nutzen diese Methode um ihre Realisierungsvision zu dokumentieren und vor der Implementierung mit den Stakeholdern zu verifizieren. Dabei bilden die Storyboards eine wesentlich bessere Diskussionsgrundlage als verbaler oder textueller Informationsaustausch. Und neben der Vermeidung von Missverständnissen bieten Storyboards den Vorteil, dass Diskussionen effizienter, präziser und schneller stattfinden können.

## Fazit

Die Kommunikation mit den Stakeholdern ist für Entwicklungs-Teams ein elementarer Schritt zum erfolgreichen Software-Projekt. Dabei müssen Software-Entwickler sich dieser Aufgabe stellen und mit geeigneten Prozessen und Hilfsmitteln diesen Austausch nach Kräften fördern, statt sich „im stillen Kämmerlein zu vergraben".

Beide Methoden zur Optimierung der Kommunikation mit Stakeholder sollen allerdings nicht die direkte Kommunikation ersetzen, sondern diese ergänzen. Jedoch ist gerade bei größeren Projekten mit einer großen Anzahl Stakeholder eine direkte Kommunikation zwischen allen Beteiligten nicht immer in der erforderlichen Intensität möglich. Hier können solche Tools eine sinnvolle Unterstützung bieten.

Siehe auch [Moderne Softwareentwicklung 3 - Stakeholder effizient einbinden]

## Continuous Delivery als zentraler Motor für erfolgreiche Projekte

Auch aus technischer Sicht bringen kürzere Zyklen „frischen Wind“ in die Software-Entwicklung, in dem die Zeit zwischen der Programmierung eines Features und der Bereitstellung für Tester und Kunden verkürzt wird. Der Fokus des ganzen Teams liegt also darin, ein Feature fix-und-fertig an den Kunden auszuliefern, anstatt viele verschiedene Dinge gleichzeitig zu starten.

Anstelle erst Tage oder gar Wochen später eine Rückmeldung zu erhalten, dass ein Feature den Test aus irgendwelchen Gründen nicht bestanden hat, erhält man ein zeitnahes Feedback, während die Erinnerung noch frisch ist.
Diese Erkenntnis ist nicht neu: Je später man ein unerwünschtes Verhalten findet, desto aufwändiger wird die Korrektur. Am schönsten wäre es natürlich, schon während des Tippens in Visual Studio darauf hingewiesen zu werden, dass die Zeile Code, die man gerade geschrieben hat, aufgrund von Abhängigkeiten etwas anderes kaputt macht.

Schnelles Feedback bekommt man (in zeitlicher Abfolge) von:

1. **IntellISense**

   Schon beim Tippen „ahnt“ Visual Studio, dass die Zeile wohl nicht kompilieren wird und hilft beim der richtigen Schreibweise – und wartet nicht bis zur Kompilierung.

2. **Compile & Link**

   Der Compiler prüft, dass alle nötigen Abhängigkeiten vorhanden sind und der Code kompiliert. Auch wenn ein Stück Code kompiliert, heißt das noch lange nicht, dass es das tut, was es tun sollte (bzw. was der Entwickler beabsichtigt hatte), deshalb…

3. **Entwickler-Test**

   … schreibt der Entwickler mit Komponententests (Unit Tests) Code um zu prüfen, ob sich der Produktivcode wie gewünscht verhält. Die Unit Tests dienen auch anderen Teammitgliedern als Sicherheitsnetz bei Änderungen am Code und als technische Dokumentation, wie sich der Code verhalten soll.

4. **Check-In**

   Durch das Einchecken werden die Änderungen des einzelnen Entwicklers mit den Änderungen des gesamten Teams zusammengeführt. Das birgt immer Potential für Konflikte, sodass man mit diesem Schritt möglichst nicht lange warten sollte.

5. **Continuous Integration (CI)**

   Nach dem Einchecken wird der Code vom Builderserver, der dem Team zentral zur Verfügung steht, nochmals in einer sauberen Umgebung gegengeprüft. Dabei wird der Code mindestens kompiliert, häufig laufen danach noch Tests oder weitere Qualitätsanalysewerkzeuge.

Man glaubt gar nicht, wie häufig der Fall auftritt, dass es auf einem Entwicklerrechner scheinbar funktioniert, der Buildserver jedoch ein Kompilierfehler meldet – zum Beispiel wegen fehlender Dateien oder Abhängigkeiten.

6. **Integration-Test (Test Lab)**

   Um herauszufinden, ob die Software überhaupt startet, kommen Integrations-Tests oder auch UI-Tests zum Einsatz. Diese erfordern eine funktionierende Testumgebung ("Test Lab"), die meistens erst recht aufwendig bereitgestellt und eingerichtet werden muss. Gerade hier lohnt sich die Automatisierung, damit die Ergebnisse zeitnah (z.B. über Nacht) ausgeführt werden und morgens eine E-Mail mit den Testergebnissen vorliegt.

7. **Testen von System-/Performance-/Technischen Anforderungen**

   Im nächsten Schritt ließen sich auf der Testumgebung auch technische Tests ausführen, die jeweils das Gesamtsystem bzw. bestimmte nicht-funktionale Aspekte wie Sicherheit, Performance, Verhalten unter Last oder besondere Situationen testen.

8. **User Acceptance Testing**

   Sind alle automatischen Tests durchlaufen, folgt das manuelle Testen. Neben dem Prüfen von Teilen, die sich nur schwer automatisieren lassen (oder noch nicht automatisiert wurden) liegt der Schwerpunkt hier auf der Erfüllung der Anforderungen. Die Leitfrage lautet, ob die Anforderung aus Sicht des Kunden erfüllt und umgesetzt wurde.

9. **Operational Acceptance Testing (Production)**

   Egal, wie viel man testet und wie ähnlich die eigenen Testumgebungen an der Produktionsumgebung angelehnt sind, es wird wohl immer Probleme geben, die erst auf der Zielumgebung auffallen. Dazu gehören auch Kennzahlen, die man verwenden kann, um das System zu überwachen und zu bemerken, wenn es sich nach einem Update anders verhält (also z.B. dass man seit dem letzten Update nur halb so viele Artikel pro Stunde verkauft).

Gerade, wenn man sehr kurze Zyklen verwendet, wie es einige Webseiten tun, sollte man hier einen „Plan B“ haben, wie man im Fehlerfall vorgeht.Und natürlich sollte man daraus lernen und ggf. die Tests erweitern, um ähnliche Fehler früher zu bemerken.

## Wann ist ein Feature eigentlich fertig?

Kunden finden kürzere Zyklen naturgemäß eher gut, geht das doch häufig mit einer kürzeren Wartezeit bis zum neuen Feature einher. Letztlich ist ein Feature erst dann fertig, wenn es fertig entwickelt, getestet und ausgeliefert wurde und es der Kunde sinnvoll bei sich nutzen kann.
Wenn man die Zeit stoppt zwischen dem Eingang einer Idee bis hin zu dem Moment, an dem das Feature vom Kunden angenommen ist, so erhält man die Durchlaufzeit (Cycle Time). Bei

Fehlerberichten wird dies als mittlere Zeit bis zur Wiederherstellung (Mean Time to Repair; MTTR) bezeichnet. Je mehr von der oben stehenden Liste automatisiert wurde, desto kürzer werden hier die Zeiten ausfallen.

Aus Entwicklersicht sollte man im Team das Wort „fertig" erst einmal definieren. Was bedeutet „fertig" für uns? Ist das inklusive Tests (auf jeden Fall!)? Mit Dokumentation? Sind Compilerwarnungen erlaubt oder ist man dann nicht „fertig"? Nur, wenn man eine explizite Definition von „fertig" bzw. „done" vorliegt, ist jedem klar, was alles zu tun ist (und was nicht). Die „Definition of Done" enthält die Mindestanforderungen, die ein Feature erfüllen muss, um als „fertig" bezeichnet zu werden – und sichert das Team auch dagegen ab, dass Dinge nach hinten geschoben oder gar übersprungen werden (denn dann wäre es nicht „fertig").
Also am besten gleich nach dem Urlaub mit dem Team zusammensetzen und eine „Definition of Done" erstellen und schriftlich festhalten.

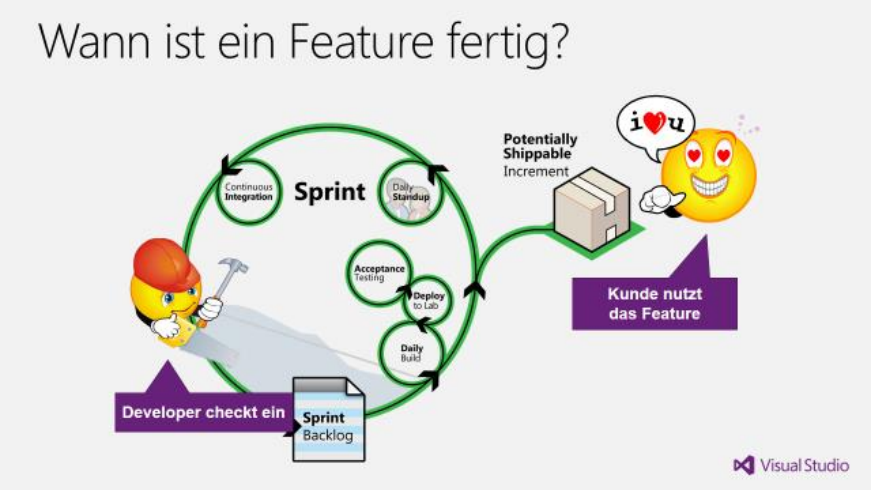



*Abbildung: Viele Schritte liegen zwischen dem Einchecken des Entwicklers und bis zum Eintreffen des neuen Features beim Kunden. Diese Zeitspanne zu verkürzen ist ein zentrales Anliegen.*

## Fazit

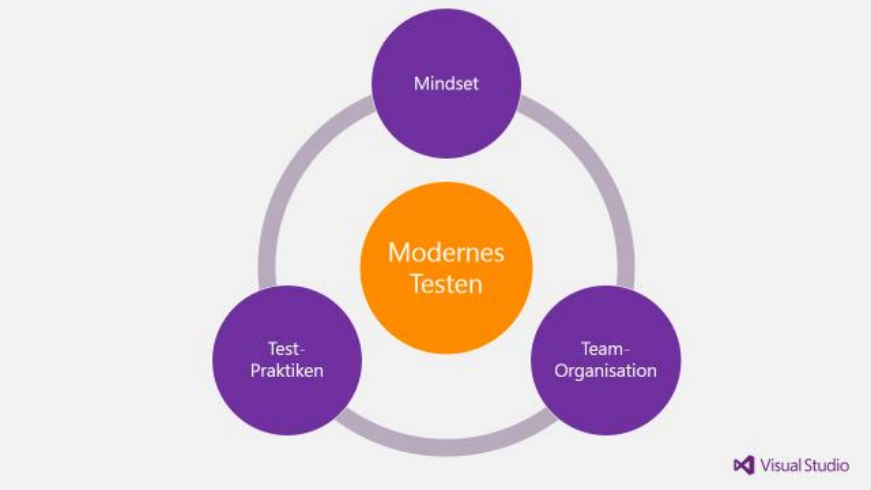
Kurze Zyklen sind möglich, erfordern aber ein hohes Maß an Automatisierung, eine funktionierende kontinuierliche Integration und Bereitstellung und eine klare Definition im Team, was eigentlich „fertig" bedeutet.

Siehe auch Moderne Softwareentwicklung 4 - Continuous Delivery als zentraler Motor für erfolgreiche Projekte

## Optimierte Qualitätssicherung für moderne Prozesse

Software-Entwicklungsteams stehen heute vor der Herausforderung, dass in immer kürzeren Zyklen Software geliefert werden soll. Gleichzeitig werden die Systeme immer komplexer und der Qualitätsanspruch der Anwender steigt. Diese Situation macht Veränderungen in Bezug auf die Qualitätssicherung erforderlich, die sich in 3 Bereiche unterteilen lassen:

1.) Das Mindset, also die Einstellung von Software-Entwicklern zu Qualität.
2.) Die Team-Organisation verbunden mit der Frage: „Wer testet wann?".
3.) Und die Testpraktiken, also „Wie können wir das Testen effizient gestalten?".




### Team-Organisation

Aktuell haben viele Unternehmen eine klare Trennung zwischen Entwicklungs- und Qualitätssicherungsteams bzw. –Abteilungen. Entwickler erstellen Code, übergeben diesen an die Tester und beheben anschließend Fehler, die der Tester findet. Jedoch bietet dieses Vorgehen eine Reihe von Nachteilen wie beispielsweise hohe Transaktionskosten (Kosten für die Kommunikation zwischen Tester und Entwickler) oder das Risiko, dass bestimmte Probleme unentdeckt bleiben, weil der Tester nicht das erforderliche Wissen über die Anwendung hat.

### Mindset

Idealerweise berücksichtigen Entwickler die Qualitätsansprüche bereits beim Implementieren des Codes. Und am effizientesten erreicht man dieses Ziel dadurch, dass es keine Tester mehr gibt. Ja, so radikal und unglaublich sich das anhört, das Entfernen von Testern aus Software-Projekten erhöht die Qualität – sofern diese Maßnahme durch eine Qualifizierung der Entwickler in Bezug auf modernes Testen begleitet wird.

Dem entgegen steht die Theorie, dass ein Entwickler nie seinen eigenen Code testen soll. Während diese Aussage sicherlich in einigen Aspekten zutrifft (z.B, Vorteile des 4-Augen-Prinzips), so dient sie Entwicklern oft als Rechtfertigung oder Ausrede, Tests nichts selbst ausführen zu müssen. Grund ist, dass Entwickler nicht gerne testen und somit auch keine hohe Motivation haben, ihre Fertigkeiten in diesem Bereich zu verbessern.

## Test-Praktiken

Jedoch haben Entwickler oftmals viel bessere Möglichkeiten, Tests effizient zu gestalten, gerade in Bezug auf Test-Automatisierung. Und letztendlich werden Entwickler auch anderen, besseren Code schreiben, wenn sie wissen, dass sie selbst die Tests dafür ausführen müssen. Einige Konzepte wie z.B. Unit-Tests oder TDD (Test Driven Development) lassen sich nicht sinnvoll umsetzen, wenn die Implementierungs- und Testaufgaben auf 2 Personen aufgeteilt sind, die unabhängig voneinander, möglicherweise in getrennten Teams arbeiten. Und auch wenn in einigen Teams Software-Entwicklung ohne dedizierte Tester nicht realisierbar erscheint, sollte dennoch versucht werden, diesem Ideal möglichst nahe zu kommen und möglichst viel der Verantwortung für die Qualität der Anwendung bereits beim Schreiben des Codes durch die Entwickler übernehmen zu lassen. Die Qualität wird also quasi in das Produkt eingebaut. Die nachgelagerten Tester sind nur noch ein Sicherheitsnetz. Sie sollten aber eigentlich keine Fehler mehr finden. Und um Transaktionskosten in diesem Szenario zu minimieren, sollten Tester auf jeden Fall Teil des Entwicklungs-Teams sein, idealerweise mit den Entwicklern in einem Raum sitzen und nicht organisatorisch getrennt sein.

Als nächsten Schritt, vor allem in Bezug auf kürzere Iterations-Zyklen, spielt die Testautomatisierung eine wichtige Rolle. Nur damit werden Teams es schaffen, in Zyklen von wenigen Tagen oder Wochen jeweils am Ende einer Iteration qualitativ hochwertige Software bereitstellen zu können. Die Testautomatisierung beginnt mit dem Schreiben von Unit-Tests. Unit-Tests gehören heute zum absoluten Muss, was Software-Entwicklung anbelangt. Und wenn es richtig gemacht wird, müssen Entwicklungsteams für das Schreiben der Unit-Tests keine zusätzliche Zeit aufwenden, da viele andere Dinge dadurch effizienter gestaltet werden können. Die effiziente Erstellung von Unit-Tests setzt allerding eine geeignete Architektur voraus – eine Voraussetzung, an der Teams mit einer großen Basis an Legacy-Code häufig scheitern.

Über Unit-Tests hinaus bieten moderne Test-Tools auch die Möglichkeit, über Integrations-Tests das Gesamtsystem effizient zu testen. So können über Virtualisierung Testumgebungen bereitgestellt werden, in denen automatische UI-Tests in möglichst realistischen Szenarien die Anwendung prüfen und so das Risiko der Übernahme von Fehlern nach dem Deployment in die Produktivumgebung

reduziert. Last- und Performance-Tests helfen, böse Überraschungen bezüglich des Laufzeitverhaltens im Produktivbetrieb zu verhindern, indem diese Probleme bereits früh im Entwicklungsprozess erkannt und beseitigt werden.

Während automatisierte Tests die Möglichkeit bieten, sehr häufig ausgeführt zu werden um Abweichungen früh zu erkennen, haben manuelle Tests nach wie vor eine wichtige Funktion. So bieten „explorative" Tests den Vorteil, dass nicht nur vordefinierte Szenarien geprüft werden, sondern die Anwendung allgemein auf unerwartetes Verhalten geprüft wird.

### Fazit

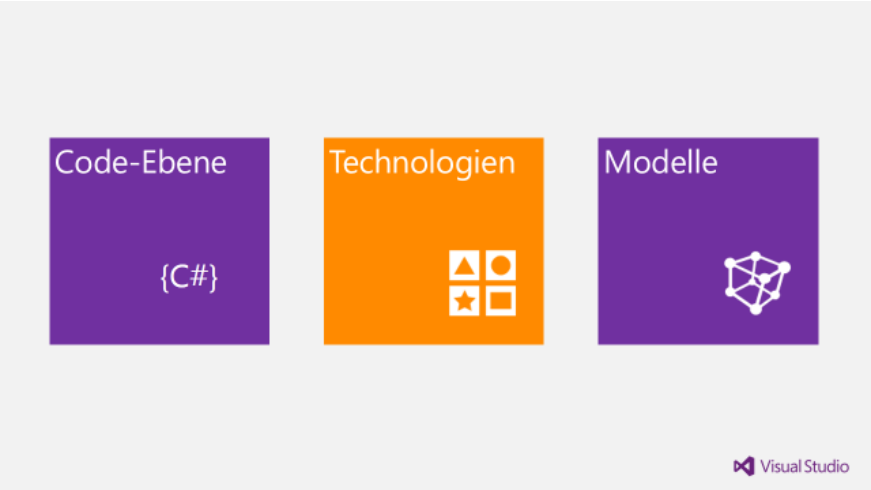
Bei der Entwicklung von Software in immer kürzeren Zyklen brauchen wir neue Methoden zur Qualitätssicherung. Dabei müssen Entwickler bereits bei der initialen Implementierung mehr Verantwortung übernehmen und die dazu erforderlichen Praktiken erlernen und anwenden. Der Lohn dafür ist eine neue Sicherheit und ein Ende des „Gezerres" zwischen Entwicklung und Qualitätssicherung vor allem am Ende eines Releases.

Siehe auch Moderne Softwareentwicklung 5 - Optimierte Qualitätssicherung für moderne Prozesse

## Erweiterbare und wartbare Architekturen entwickeln

Klassische Architektur liefert meistens spezialisierte Lösungen für konkrete Probleme oder Anforderungen. Deshalb ist der klassische Wasserfall-Prozess auch so aufgebaut, dass zunächst alle Anforderungen bekannt sein müssen, um dann die ideale Architektur zu entwerfen. Das bedeutet aber auch, wenn sich die Anforderungen ändern, kann es vorkommen, dass das Architekturkonzept nicht mehr passt.

Im Gegensatz dazu funktioniert moderne Architektur mehr mit allgemeinen Patterns, die sich auf verschiedene Projekte und Situationen anwenden lassen. Diese Patterns muss jeder Entwickler beherrschen, da er sie in seiner täglichen Entwicklungsarbeit anwenden muss; d.h. Architektur manifestiert sich in diesem Zusammenhang sehr stark direkt in der Code-Ebene. Zusätzliche Aspekte von Architektur betreffen Technologien und Modelle.




Die Code-Ebene bezieht sich auf Strukturen im Code und bietet Lösungspatterns für Fragen wie z.B.

- Wie strukturiere ich meine Klassen?
- Welche Abhängigkeiten bestehen zwischen verschiedenen Code-Teilen?
- Wie gestalte ich Schnittstellen zwischen meinen Komponenten?

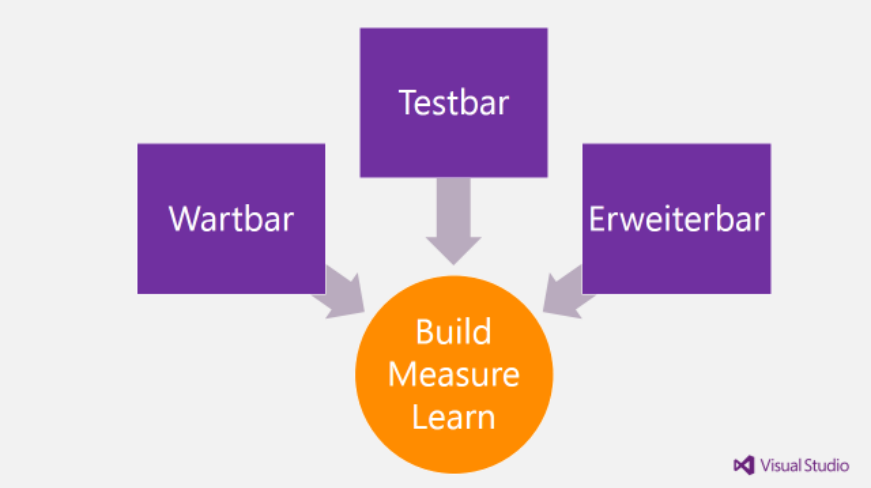
Bei der Technologie-Ebene gibt es in Bezug auf die Architektur zwei Aspekte. So ermöglichen Technologien auf der einen Seite gewisse Architektur-Patterns erst. Beispiele dafür sind WPF, wodurch das MVVM-Pattern erst richtig elegant umsetzbar wurde, oder WCF und Webservices, die serviceorientierte Architekturen zum Durchbruch verholfen haben. Auf der anderen Seite muss Architektur es auch ermöglichen, Technologien auf enge Bereiche des Codes zu beschränken, sodass beim  Wechseln einer Technologie die notwenigen Anpassungen auf diesen Bereich begrenzt werden können.

Modelle stellen die Visualisierung von Architektur dar. Dies ist wohl die heute gängigste Assoziation mit dem Begriff Architektur. Beim klassischen Ansatz werden Modelle genutzt, damit ein Spezialist (der Software-Architekt) seine Konzepte und Ideen an die Entwickler kommunizieren kann, die diese in Code umsetzen sollen. Moderne Teams kommen jedoch ohne klassischen Architekten aus, da das Team gemeinsam Verantwortung für die Architektur übernimmt und jeder im Team aktiv an deren Gestaltung mitwirkt. Gerade hier erfüllen Modelle einen wichtigen Aspekt, nämlich die Visualisierung von Strukturen und die Reduktion von Code auf verschiedene Abstraktionsebenen. Dies erlaubt eine wesentlich effizientere Kommunikation im Team über Architektur.

Der klassische Architekt wird durch diese Veränderung jedoch nicht arbeitslos, sondern er übernimmt eine neue wichtige Aufgabe. Er übernimmt die Aufgabe eines Coaches für die Entwickler und sorgt dafür, dass die Entwickler die notwendigen Patterns kennen und korrekt anwenden können. Und er

entdeckt und entwickelt neue Patterns, da es hier ja auch kontinuierliche Verbesserungen und Weiterentwicklungen gibt und transferiert dieses neue Wissen nach einer Evaluierung an das Team weiter.

Moderne Architektur muss drei wichtige Aspekte erfüllen:




Die **Testbarkeit** ermöglicht es uns, dass wir effizient Unit-Tests mit klaren und atomaren Testzielen erstellen können. Dazu hilft z.B. auch Entkopplung, die es uns u.a. ermöglicht, bestimmte Teile unserer Anwendung durch sog. Mockups zu ersetzen, welche diesen Teil simulieren (z.B. den Zugriff auf eine Datenbank), ohne für die Testausführung eine Datenbank mit entsprechenden Test-Daten bereitstellen zu müssen.

**Wartbarkeit** beschreibt das Ziel, Veränderungen an der Anwendung schnell und vor allem ohne das Risiko, andere Bereiche zu beeinträchtigen, durchführen zu können. Die Wartbarkeit erlaubt es z.B. auch, dass Fehler durch alle Entwickler im Team behoben werden können, unabhängig davon, wer den Code ursprünglich geschrieben hat. Wartbarkeit beschleunigt auch die Fehlersuche und ermöglicht es, die notwendigen Änderungen möglichst lokal zu begrenzen.

Durch **Erweiterbarkeit** bekommen wir die Möglichkeit, neue Funktionen schnell und effizient zu unserer Anwendung hinzuzufügen. In jeder Anwendung steigt der Aufwand, neue Funktionalität einzubauen mit der Gesamtmenge an Code an. Diesen Anstieg möglichst gering zu halten ist das primäre Ziel von Erweiterbarkeit.

Alle diese Aspekte stellen eine Voraussetzung für ein effizientes Build-Mesure-Learn dar. Testbarkeit ermöglicht das Liefern qualitativ hochwertigen Codes in kurzen Zyklen, Wartbarkeit ermöglicht

schnelle Anpassung bestehender Funktionalität und Erweiterbarkeit erlaubt es, Anwenderwünsche, die aus unserem Learn-Schritt entstehen, schnell und effizient umzusetzen.

## Fazit

Während der Erfolg der Bemühungen in Bezug auf Wartbarkeit und Erweiterbarkeit sich nur mittel- bis langfristig messen lässt, bietet die Testbarkeit den Vorteil, dass sie sofort überprüfbar ist – schon in dem Moment, in dem ich Tests dafür schreibe. Schön ist an dieser Stelle nun, dass dieselben Patterns, die die Testbarkeit verbessern auch zu einer Verbesserung der Erweiterbarkeit und Wartbarkeit führen. Wir haben hier also nicht nur kein Problem mit konkurrierenden Patterns, sondern können über die Testbarkeit die Einhaltung dieser Patterns auch sehr kurzfristig überprüfen.

Siehe auch Moderne Softwareentwicklung 6 - Erweiterbare und wartbare Architekturen im Team entwickeln

## Weiterführende Literatur

- The Lean Startup von Eric Ries
- Visual Studio Team Foundation Server 2012: Adopting Agile Software Practices von Sam Guckenheimer (Microsoft) und Neno Loje
- Agile Principles, Patterns and Practices in C# von Robert C. Martin

## Weiterführende Trainings

- Professional Scrum Developer (eine Initiative von Microsoft und scrum.org)

## Thomas Rümmler (AIT GmbH) – Wartung und Versionierung von Process Templates

### Arbeitgeber

• AIT GmbH & Co. KG

• ALM Consultant

### Technologieschwerpunkte

• Entwicklungsprozesse mit dem TFS unterstützen

• Large Scale

• Project Management

### Mein Urlaubstipp

Mein Lieblingsort zum Entspannen ist dort, wo meine Gitarre ist. Sie bringt mich auf andere Gedanken ganz unabhängig vom Wetter oder anderen äußeren Einflüssen. Ein paar vertraute Handgriffe genügen, um jeden Ort der Welt zu nutzen, um zu relaxen und dabei vielleicht sogar noch den einen oder anderen vernünftigen Ton aus den Saiten zu holen. Solch einen „mobilen Urlaubsort“ kann ich nur empfehlen. Wenn man dann entspannt hat kann man auch gleich zur Insellektüre greifen und die Urlaubsstimmung ist perfekt.

### Meine Inseltechnologie

Eines der spannendsten Themen des letzten Jahres im ALM Umfeld war die Integration des Team Foundation Servers in bestehende Landschaften. Dabei haben beschriebene Prozesse ebenso Einfluss genommen auf die Einführung des TFS wie vorhandene Toollandschaften. Sei man in einer heterogenen Umgebung mit Windows- und Linux-Entwicklung oder in einem streng regulierten Bereich mit speziellen Anforderungen an die Archivierung, in einem extrem volatilen Markt mit kurzen Releasezyklen oder in einer eher schwergewichtigen Produktentwicklungslandschaft. Der TFS als ALM-

Plattform hat sich unter all diesen Umständen durchgeschlagen, mal mit mehr und mal mit weniger intensiven Anpassungen.

## Wartung und Versionierung von Process Templates

Den Workflow oder einzelne Prozessschritte in der Softwareentwicklung anzupassen, sind mit dem Team Foundation Server relativ leicht möglich. Man muss nur das Process Template anpassen und hat im Handumdrehen ein neues Feld zum Task hinzugefügt oder andere kleine Erweiterungen gemacht. Dabei wird der Nutzer auch durch verschiedene Tools, wie z.B. den Team Foundation Server Power Tools sehr gut unterstützt. Doch wie bewahrt man das Process Template vor unnötigen Anpassungen und wie sieht es eigentlich mit der Versionierung dieser Änderungen aus? Woher weiß man, welchem Team Project im TFS welches Process Template in welcher Version des Process Templates zugrunde liegt?

Oberstes Gebot bei der Bearbeitung von Process Templates ist die Verwendung der Versionskontrolle. Bevor man eine Anpassung macht, sollte man den Source Code eines Process Templates zunächst in seiner noch unveränderten Form in einem TFS Repository einchecken. Dies ermöglicht die Vorteile einer versionssicheren Dateiablage zu nutzen, um später Änderungen besser nachvollziehen oder Stände miteinander vergleichen zu können. Wenn man häufig Anpassungen der Art macht, dann ergibt sich in der Versionskontrolle in regelrechtes Sammelsurium an Process Templates (siehe Abbildung 1).

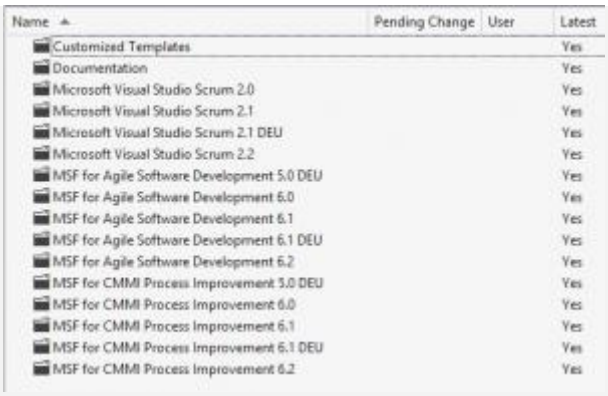



*Abbildung 1: Sammelsurium an Process Templates*

Damit hat man einen sehr wichtigen Grundstein für die Weiterentwicklung gelegt, auf dem man stabil und verlässlich weiter an seinem Template arbeiten kann. Dies ist jedoch nur der erste Schritt. Direkt als nächstes ist es wichtig, ein haltbares Versionierungsschema zu implementieren.

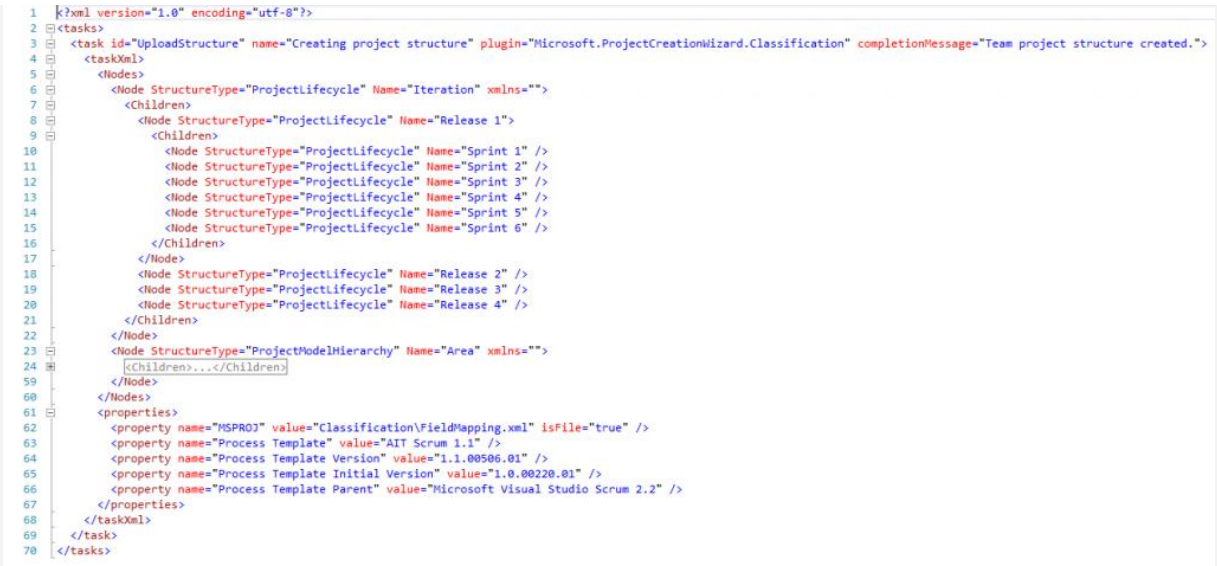
Nachdem ein neues Team Project auf Basis eines bestimmten Process Templates im Team Foundation Server erstellt wurde, kann man nämlich nicht mehr so einfach herausfinden, welches Process Template beim Anlegen des Team Projects verwendet wurde. Mit der Version 2012 des Team Foundation Servers hat Microsoft hier nachgebessert. Die Datei Classification.xml des Process Templates ist in der Lage eigene Properties aufzunehmen.

Genau dieses Feature (übrigens auch schon in TFS 2010 verfügbar) hat Microsoft nun erstmals selbst genutzt, um ein Property namens "Process Template" einzufügen (siehe dazu Abbildung 2). In dem Screenshot ist die Datei *Classification.xml* des von Microsoft ausgelieferten Process Templates *Microsoft Visual Studio Scrum 2.2* zu sehen. Die in Zeile 25 beginnende Sektion *properties* gab es bereits vor dem Team Foundation Server 2012. Nun hat Microsoft diese jedoch selbst genutzt, um Versionsinformationen zu hinterlegen. Diese werden beim Erstellen eines neuen Team Projects mit in der TFS Datenbank gespeichert und sind somit später abrufbar (später mehr dazu).

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <tasks>
3    <task id="UploadStructure" name="Creating project structure" plugin="Microsoft.ProjectCreationWizard.Classification" completionMessage="Team project structure created.">
4      <taskXml>
5        <Nodes>
6          <Node StructureType="ProjectLifecycle" Name="Iteration" xmlns="">
7            <Children>
8              <Node StructureType="ProjectLifecycle" Name="Release 1">
9                <Children>
10                 <Node StructureType="ProjectLifecycle" Name="Sprint 1" />
11                 <Node StructureType="ProjectLifecycle" Name="Sprint 2" />
12                 <Node StructureType="ProjectLifecycle" Name="Sprint 3" />
13                 <Node StructureType="ProjectLifecycle" Name="Sprint 4" />
14                 <Node StructureType="ProjectLifecycle" Name="Sprint 5" />
15                 <Node StructureType="ProjectLifecycle" Name="Sprint 6" />
16               </Children>
17             </Node>
18             <Node StructureType="ProjectLifecycle" Name="Release 2" />
19             <Node StructureType="ProjectLifecycle" Name="Release 3" />
20             <Node StructureType="ProjectLifecycle" Name="Release 4" />
21           </Children>
22         </Node>
23         <Node StructureType="ProjectModelHierarchy" Name="Area" xmlns="" />
24       </Nodes>
25       <properties>
26         <property name="MSPROJ" value="Classification\FieldMapping.xml" isFile="true" />
27         <property name="Process Template" value="Microsoft Visual Studio Scrum 2.2" />
28       </properties>
29     </taskXml>
30   </task>
31 </tasks>
```

*Abbildung 2: Classification.xml des Scrum 2.2 Templates*

An dieser Stelle kann man nun ansetzen und auf die gleiche Weise weitere Eigenschaften bekanntmachen. Zwei zusätzliche Informationen, die sich als nützlich erwiesen haben, sind die Versionsnummer, unter der das Template initial verwendet wurde sowie ein Verweis auf das Process Template, welches als Basis für die eigene Weiterentwicklung gedient hat. Die konkrete Vergabe der Versionsnummer kann natürlich frei gewählt werden. In dem hier gezeigten Beispiel ist das AIT-Versionsnummernschema zum Einsatz gekommen. Ein möglicher Aufbau der Datei Classification.xml nach diesem Schema ist in Abbildung 3 dargestellt.



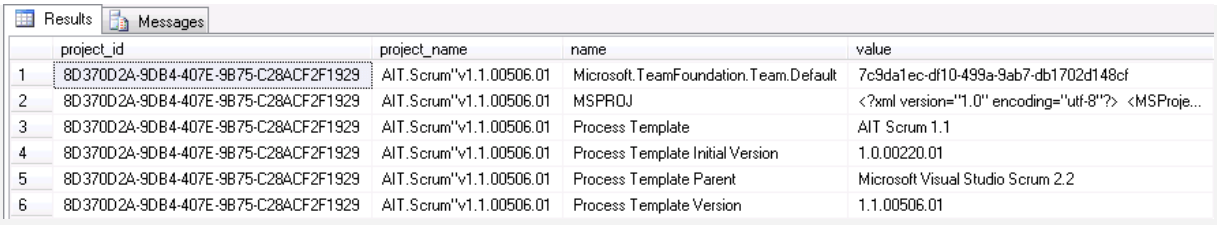
```
<?xml version="1.0" encoding="utf-8"?>
<tasks>
  <task id="UploadStructure" name="Creating project structure" plugin="Microsoft.ProjectCreationWizard.Classification" completionMessage="Team project structure created.">
    <taskXml>
      <Nodes>
        <Node StructureType="ProjectLifecycle" Name="Iteration" xmlns="">
          <Children>
            <Node StructureType="ProjectLifecycle" Name="Release 1">
              <Children>
                <Node StructureType="ProjectLifecycle" Name="Sprint 1" />
                <Node StructureType="ProjectLifecycle" Name="Sprint 2" />
                <Node StructureType="ProjectLifecycle" Name="Sprint 3" />
                <Node StructureType="ProjectLifecycle" Name="Sprint 4" />
                <Node StructureType="ProjectLifecycle" Name="Sprint 5" />
                <Node StructureType="ProjectLifecycle" Name="Sprint 6" />
              </Children>
            </Node>
            <Node StructureType="ProjectLifecycle" Name="Release 2" />
            <Node StructureType="ProjectLifecycle" Name="Release 3" />
            <Node StructureType="ProjectLifecycle" Name="Release 4" />
          </Children>
        </Node>
        <Node StructureType="ProjectModelHierarchy" Name="Area" xmlns="">
          <Children>...</Children>
        </Node>
      </Nodes>
      <properties>
        <property name="MSPROJ" value="Classification\FieldMapping.xml" isFile="true" />
        <property name="Process Template" value="AIT Scrum 1.1" />
        <property name="Process Template Version" value="1.1.00506.01" />
        <property name="Process Template Initial Version" value="1.0.00220.01" />
        <property name="Process Template Parent" value="Microsoft Visual Studio Scrum 2.2" />
      </properties>
    </taskXml>
  </task>
</tasks>
```

*Abbildung 3: Classification.xml eines angepassten Process Templates*

Wie bereits angekündigt, ist es ein Leichtes, diese Informationen wieder aus einem existierenden Team Project auszulesen. Dafür gibt es verschiedene Möglichkeiten. Man kann sich z.B. ein kleines Tool schreiben, welches über die TFS API diese Informationen zur Verfügung stellt. Eine Alternative ohne die TFS API ist der Zugriff auf die TFS Datenbank mit einem einfachen Select-Statement.

```
SELECT [tbl_projects].[project_id]
  , [tbl_projects].[project_name]
  , [tbl_project_properties].[name]
  , [tbl_project_properties].[value]
FROM [tbl_projects]
INNER JOIN [tbl_project_properties]
 ON [tbl_projects].[project_id] = [tbl_project_properties].project_id
WHERE [tbl_projects].[project_name] like 'AIT.Scrum%'
```

Diese Abfrage kann einfach wiederverwendet werden. Man muss lediglich den Namen des Team Projects in der Where-Clause austauschen. Abbildung 4 zeigt das Ergebnis der Abfrage.



Results | Messages

| | project_id | project_name | name | value |
|---|---|---|---|---|
| 1 | 8D370D2A-9DB4-407E-9B75-C28ACF2F1929 | AIT.Scrum''v1.1.00506.01 | Microsoft.TeamFoundation.Team.Default | 7c9da1ec-df10-499a-9ab7-db1702d148cf |
| 2 | 8D370D2A-9DB4-407E-9B75-C28ACF2F1929 | AIT.Scrum''v1.1.00506.01 | MSPROJ | <?xml version=''1.0'' encoding=''utf-8''?> <MSProje... |
| 3 | 8D370D2A-9DB4-407E-9B75-C28ACF2F1929 | AIT.Scrum''v1.1.00506.01 | Process Template | AIT Scrum 1.1 |
| 4 | 8D370D2A-9DB4-407E-9B75-C28ACF2F1929 | AIT.Scrum''v1.1.00506.01 | Process Template Initial Version | 1.0.00220.01 |
| 5 | 8D370D2A-9DB4-407E-9B75-C28ACF2F1929 | AIT.Scrum''v1.1.00506.01 | Process Template Parent | Microsoft Visual Studio Scrum 2.2 |
| 6 | 8D370D2A-9DB4-407E-9B75-C28ACF2F1929 | AIT.Scrum''v1.1.00506.01 | Process Template Version | 1.1.00506.01 |

*Abbildung 4: Ergebnis der SQL-Abfrage*

Die Process Templates, die mit dem TFS 2012 ausgeliefert werden, enthalten noch eine weitere Neuerung bzgl. der Versionierung. In der Root Process Template Datei ProcessTemplate.xml gibt es bei den Metadaten des Templates (XML-Tag metadata) das neue version-XML-Tag (siehe Abbildung 5). Dort erhält das Process Template einen eindeutigen Identifier sowie eine interne, zweigeteilte Versionsnummer in Form eines Major- und Minorteils.

```
<?xml version="1.0" encoding="utf-8"?>
<ProcessTemplate>
  <metadata>
    <name>Microsoft Visual Studio Scrum 2.0</name>
    <description>This template is for teams who follow the Scrum methodology and use Scrum terminology.</description>
    <version type="6B724908-EF14-45CF-84F8-768B5384DA45" major="2" minor="20"/>
    <plugins>
      <plugin name="Microsoft.ProjectCreationWizard.Classification" wizardPage="false" />
      <plugin name="Microsoft.ProjectCreationWizard.Reporting" wizardPage="false" />
      <plugin name="Microsoft.ProjectCreationWizard.Portal" wizardPage="true" />
      <plugin name="Microsoft.ProjectCreationWizard.Groups" wizardPage="false" />
      <plugin name="Microsoft.ProjectCreationWizard.WorkItemTracking" wizardPage="false" />
      <plugin name="Microsoft.ProjectCreationWizard.VersionControl" wizardPage="true" />
      <plugin name="Microsoft.ProjectCreationWizard.TestManagement" wizardPage="false" />
      <plugin name="Microsoft.ProjectCreationWizard.Build" wizardPage="false" />
      <plugin name="Microsoft.ProjectCreationWizard.Lab" wizardPage="false" />
    </plugins>
  </metadata>
```

*Abbildung 5: Process Template.xml mit version-Tag*

Diese Daten tauchen im Bereich der Process Templates auch in der TFS Datenbank wieder auf. Man kann diese Informationen nutzen, um die Version eines Process Templates im Process Template Store, also noch bevor ein Team Project erzeugt worden ist, genauer zu beschreiben.

Nachdem der Quellcode des Process Templates nun unter vollständiger Kontrolle der Quellcodeverwaltung ist und das Process Template auch noch ein paar Versionsinformationen erhalten hat, kann man den nächsten Schritt in der Weiterentwicklung erschließen: Branch-Struktur der Process Templates.

Die Tatsache, dass der Source Code durch die vorher beschriebenen Schritte sowieso im Source Control des TFS verfügbar ist, kann nun noch durch Verwendung von Branches ausgenutzt werden. Das Elternelement eines Branch-Baums ist dabei stets das vom Hersteller ausgelieferte Process Template. In dem Beispiel des Scrum Templates von Microsoft ist *Microsoft Visual Studio Scrum 2.2* also das Wurzelelement eines Branch-Baums (siehe Abbildung 6).

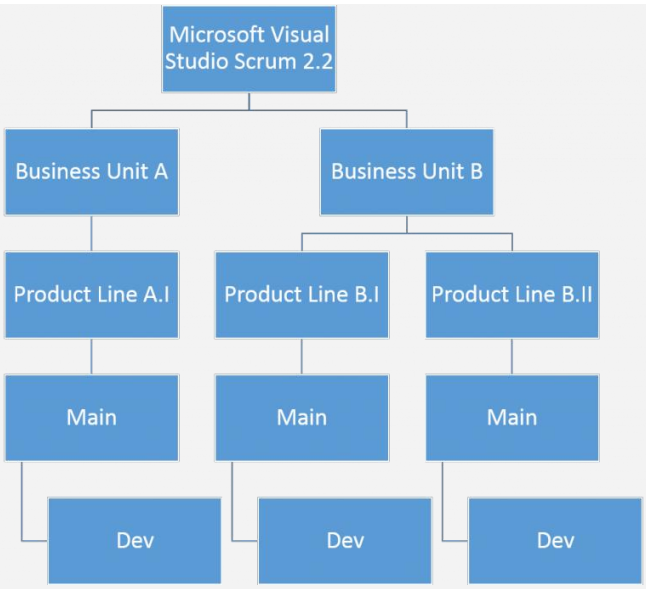


*Abbildung 6: Branch-Hierarchie*

In dem in der Abbildung dargestellten Beispiel gibt es im Unternehmen zwei verschiedene Business Units, die für die Entwicklung verschiedener Produktlinien verantwortlich sind. Dabei verwenden sie unterschiedliche Prozesse, was den Einsatz verschiedener Process Templates im TFS nach sich zieht. Die Entwicklung der beiden Produktlinien B.I und B.II ist bzgl. der Zustände der einzelnen Work Items jedoch gleich. Sie unterscheiden sich nur auf Feldebene der einzelnen Work Item Typen. Deshalb ist es sinnvoll, für diese Process Templates noch einmal einen gemeinsamen Knoten in der Branch-Hierarchie abzubilden, hier: Business Unit B.

In der Praxis hat sich gezeigt, dass es nur schwer möglich ist, Anpassungen, die innerhalb eines Process Templates entstanden sind, z.B. bei Product Line B.II, durch Reverse Integration auf andere Zweige zu verteilen. Jedoch dokumentiert die Branch-Struktur hervorragend die Entstehung sowie die Beziehungen der einzelnen Anpassungen. Außerdem kann man Änderungen allgemeiner Art relativ gut durch Forward Integration verteilen.

Hierbei ist es wie in der Softwareentwicklung entscheidend, dass die Branch-Hierarchie sorgfältig definiert wurde. In dem Beispiel aus Abbildung 5 ist es durchaus denkbar, zwischen dem Wurzelknoten und den beiden Business Units noch einen gemeinsamen Zwischenknoten einzufügen, um Änderungen, die für alle Templates zutreffen, an einer Stelle implementieren zu können. Ein Beispiel dafür kann ein Feld sein, welches in einem bestimmten Work Item Type (z.B. Task) in allen Templates vorhanden sein soll.
Wenn man diese Forward Integration zum Verteilen von Neuerungen weiter durchdenkt, kommt man noch zu einem weiteren Knackpunkt: die Formatierung des XML-Codes. Wenn man Process Template

Anpassungen im großen Stil macht, kann es sinnvoll sein, sich auf bestimmte Richtlinien zur XML-Formatierung zu einigen, um Vergleichs- und Merge-Operationen zu vereinfachen oder überhaupt erst sinnvoll zu ermöglichen.

Deutlich wird dies durch folgendes Beispiel: Einem Work Item Type (z.B. Product Backlog Item des Scrum Templates) soll ein neues Feld hinzugefügt werden. Dafür sind mindestens zwei Stellen zu bearbeiten. Zunächst muss das neue Feld innerhalb des XML-Tags *Fields* bekannt gemacht werden. Hier kommt die erste Herausforderung, denn an welcher Stelle innerhalb der genannten XML-Sektion die Felddefinition eingefügt wird, ist aus Sicht des TFS irrelevant. Jedoch ist es für einen Vergleich zweier Dateien unabdingbar, dass man sich auf eine einheitliche Vorgehensweise geeinigt hat. Dies könnte z.B. so aussehen, dass neue Felder stets unten in dem Bereich *Fields* in alphabetischer Reihenfolge angefügt werden. Eine Alternative ist, die Felddefinitionen so zu sortieren, dass sie mit der Reihenfolge der Elemente an der Oberfläche übereinstimmen.

Die zweite Schwierigkeit in diesem Umfeld ist, dass die XML-Formatierung einheitlich ist. Darunter fallen Punkte wie “Tabulator oder Leerzeichen zum Einrücken” oder die Verwendung von Kommentaren. Auch hier wird die Weiterentwicklung durch eine Vereinheitlichung erleichtert.

Wie den vorherigen Erläuterungen zu entnehmen ist, entpuppt sich die nachhaltige Pflege und Weiterentwicklung von Prozess Templates als etwas komplexer als man vielleicht auf den ersten Blick meint. Deshalb ist es umso wichtiger auch die Änderungswünsche der Nutzer kritisch zu hinterfragen und wie in anderen Projekten ein sorgfältiges Requirements Management zu leben. Eine Möglichkeit, die Process Templates stabiler zu halten, ist die Verwendung der Tags, die seit dem Update 2 (TFS 2012.2) zur Verfügung stehen. Damit kann die Anzahl der Änderungen deutlich reduziert werden. Denn alle Anfragen nach weiteren Feldern, um zusätzliche Metainformationen zu speichern, um dann beispielsweise nach weiteren Kriterien auszuwerten, können nun von den Benutzern selbst gelöst werden.

## Fazit

Process Template Customization ist sehr ähnlich zu einem Softwareentwicklungsprojekt. Es gilt zunächst, die Anforderungen gründlich zu sondieren. Darüber hinaus müssen gewisse Qualitätskriterien eingehalten werden. Anpassungen des Process Templates müssen auch getestet werden. Schließlich müssen Versionen oder Releases definiert werden, die geordnet veröffentlicht und installiert werden.

## Verwandte Artikel

- Neue Version: Dependency Manager v2.0 für TFS 2012
- Visual Studio und TFS 2012 Update 2 verfügbar

Benötigen Sie Unterstützung bei der Software-Entwicklung und Architektur von .NET basierten Lösungen oder bei Einführung und Anpassung von Visual Studio / Microsoft Test Manager / Team Foundation Server?
Wir stehen Ihnen unter info@aitgmbh.de gerne zur Verfügung.

## Sven Hubert  (AIT GmbH) – Zeiterfassung mit TFS (ohne Microsoft Project)

### Arbeitgeber

• AIT GmbH & Co. KG

• TeamSystemPro Team

### Technologieschwerpunkte

• ALM Assessment

• Prozesseinführung und -anpassung

• Agile Transition

• Toolchain Modelling

• TFS Anpassung und Einführung, Trainings und Coaching

### Mein Urlaubstipp

Wer rastet der rostet! Nichts ist erholsamer als die Abwechlsung aus Ruhe und Sport. Daher auch im Urlaub mal eine Wander- oder Fahrradtour oder einfach nur ausgedehnte Morgenspaziergänge am Strand einplanen...

### Meine Inseltechnologie

Zeiterfassung mit TFS: Damit nach dem Urlaub schnell berichtet werden kann, wie viel das Team in der Zwischenzeit getan hat und auch noch nachträglich vergessene Zeiten berichtet werden können, sind im Team Foundation Server einige Anpassungen nötig. Welche? Weiterlesen...

## Zeiterfassung mit TFS (ohne Microsoft Project)

Wenn man in TFS schon Zeitaufwände wie etwa Completed und Remaining Work erfasst und es ein Benutzermanagement gibt, kann man die Daten dann nicht auch gleich als Ersatz für die Zeiterfassung nutzen?

In diesem Teil der Blog-Reihe zu Zeiterfassung mit TFS zeigen wir Ihnen wie weit man mit TFS-Boardmitteln tatsächlich kommt…

### Was wir benötigen…

Zunächst einmal gibt es organisatorische Herausforderungen. Wenn man die Zeiterfassung durch TFS ablösen und kein weiteres System verwenden möchte, dann müssten auch:

1. alle anderen Mitarbeiter (Nicht-Entwickler/Tester) über TFS Zeiten erfassen und
2. alle Aufgaben, also auch die außerhalb des aktuellen Projektes sowie für Meetings usw. im TFS erfasst werden – und dazu pro Beteiligtem je Aufgabe je ein Task Work Item.

Das ist sicher nicht erwünscht. Doch schon die Auswertung der "Entwickler- und Testeraufwände" könnte interessant genug sein. Dazu reicht es aber nicht, am Ende des Projektes eine Gesamtsumme aus allen Tasks zu ermitteln. Die Aufwände sollen z.B. monatsweise oder gar tagesgenau pro Mitarbeiter ausgewertet werden können, um Sie z.B. mit dem Zeiterfassungssystem abzugleichen oder weitere Details über die angefallenen Tätigkeiten zu geben.

### Was der TFS-Cube bietet…

Team Foundation Server bietet Out-Of-The-Box in den Process Templates MSF for CMMI und Agile im Task Work Item die Felder Completed Work und Remaining Work an. Hier ein Beispiel aus dem Template MSF for CMMI 6:

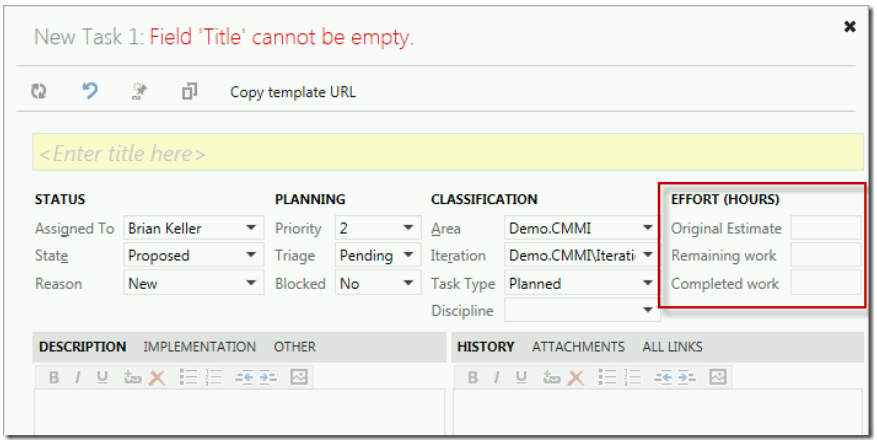



*Abbildung: Beispiel-Task aus dem Process Template MSF for CMMI Process Improvement 6*

Diese werden so verwendet, als dass in Completed Work stets die Summe aller angefallenen Aufwände und in Remaining Work die Summe der wahrscheinlich verbleibenden Aufwände eingetragen ist. Dafür muss der Benutzer sorgen. Original Estimate kann verwendet werden, um die initial geschätzte Zeit festzuhalten.

Ein Beispiel: Entwickler Dave trägt in einem neuen Task der ihm zugewiesen ist initial seine Schätzung in Original Estimate und den gleichen Wert in Remaining Work ein. Im weiteren Verlauf verwendet er nur noch die Felder Completed und Remaining Work:

| **Aktion** | **Original Estimate** | **Remaining Work** | **Completed Work** |
|---|---|---|---|
| Initiale Schätzung | 20 | 20 | |
| Erste Aufwände | 20 | 15 | 5 |
| Korrektur der Schätzung nach weiterer Arbeit | 20 | 14 | 8 |
| Abschluss der Aufgabe | 20 | | 24 |

*Tabelle: Beispiel der Änderungen an den Aufwandsfeldern im Task Work Item*

Man stellt fest, dass sowohl der Bearbeiter als auch der Auswertende ständig rechnen müssen, um zu ermitteln, wer wann wie viel gearbeitet hat. Zudem zeigt der Task ja immer nur die aktuellen Werte. Der Verlauf ist in der Historie unter allen anderen Änderungen "vergraben":

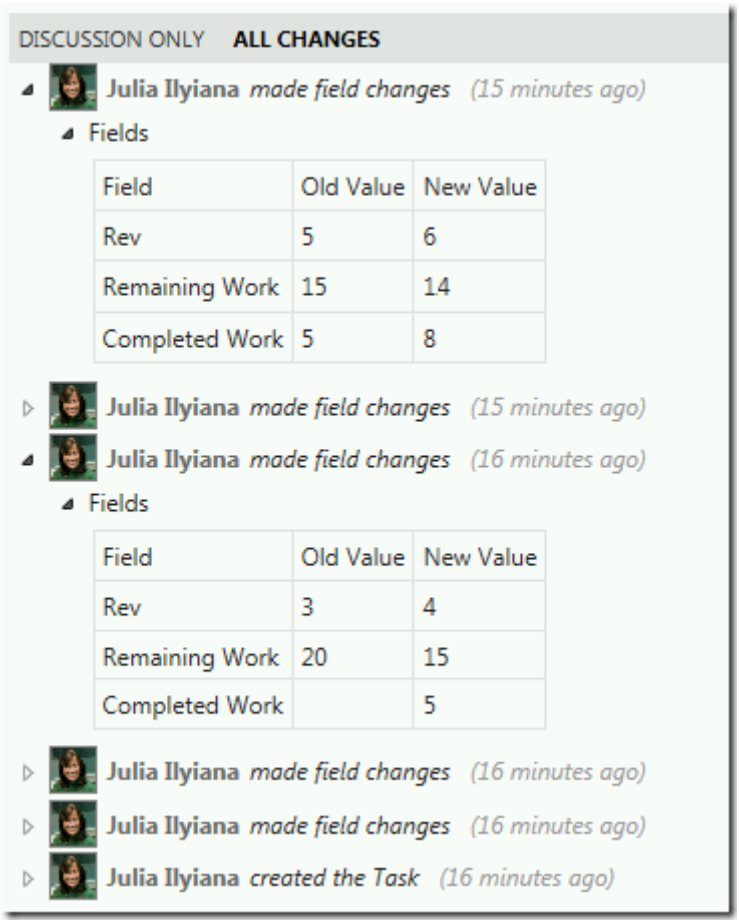



*Abbildung: Historie des Beispiel-Task*

TFS bietet mit dem Reporting (über ein Warehouse) die Möglichkeit für historische Auswertungen und die Betrachtung des Verlaufs von Werten. Wenn man über den TFS Cube geht (siehe auch Reporting mit dem TFS Teil 1/3 – Berichte mit Excel) erhält man auch recht einfach über eine Pivot-Tabelle die folgende Auswertung für die Aufwandsfelder nach Work Item Id (hier 4):

| | 3/11/2013 | | |
|---|---|---|---|
| **Row Labels** | **Original Estimate** | **Remaining Work** | **Completed Work** |
| 4 | 20,0 | 24,0 | 0,0 |
| **Grand Total** | **20,0** | **24,0** | **0,0** |

*Abbildung: Reporting der Aufwandsfelder für Task #4 am 11.03.2013*

Diese ließen sich auch mit dem Assigned-To-User verknüpfen, so dass man die Zuordnung der Tasks zu den Benutzern in der Tabelle auflisten kann.
Wenn man nun aber auch Änderungen an diesem Tag und von unterschiedlichen Benutzern darstellen möchte und deshalb die Work Item Revision mit einblendet, erhält man lediglich Nullwerte wo eigentlich nach der obigen Tabelle andere Werte stehen müssten:

| Row Labels | Original Estimate | Remaining Work | Completed Work |
|---|---|---|---|
| 4 | 20,0 | 24,0 | 0,0 |
| 2 | 0,0 | 0,0 | |
| 3 | 0,0 | 0,0 | 0,0 |
| 4 | 0,0 | 0,0 | 0,0 |
| 5 | 0,0 | 0,0 | 0,0 |
| 6 | 20,0 | 24,0 | 0,0 |
| Grand Total | 20,0 | 24,0 | 0,0 |

*Abbildung: Reporting mit Revisionsnummern des Task #4 am 11.03.2013*

Der Grund für die Nullwerte ist technisch begründet: Das Analysis-Reporting löst die Änderungen im Cube nicht auf Revisionen auf, bzw. werden Measures (aggregierbare Werte im Cube wie z.B. Completed Work) für Zwischenrevisionen pro Tag auf 0 gesetzt. Nur die letzte Änderung an einem Tag enthält tatsächlich einen Wert für ein Measure. Das liegt daran, dass die Gesamtsumme pro Work Item stimmen muss. Für eine Id muss die Summe von Completed Work pro Tag passen. Wenn nun aber pro Revision ein anderer Wert steht, würde die Gesamtsumme in Completed Work der Summe aller Revisionen entsprechen und das wäre falsch. Daher sieht man im Cube für die Revisionen immer Nullwerte.

## Wie es mit dem relationalen Warehouse besser geht…

Neben dem Cube, der für langfristige historische Auswertungen optimiert ist, gibt es ein relationales Warehouse. Im Gegensatz zum Cube lässt sich das in Excel nicht direkt in einer Pivot-Tabelle öffnen, sondern bedarf ein paar mehr Handgriffen.
Um das relationale Warehouse zu verwenden, benutzt man in Excel ebenfalls den Verbinden-Dialog:

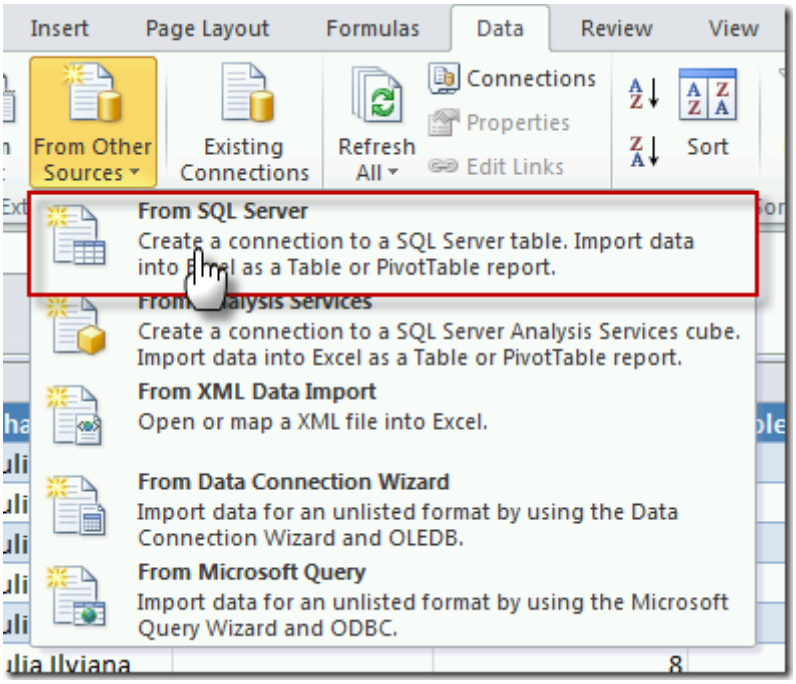



*Abbildung: Verbinden mit dem relationalen Warehouse*

Als Server gibt man das TFS Data-Tier an und wählt dann die Datenbank "Tfs_Warehouse" und darin die Tabelle "WorkItemHistoryView". Dies erzeugt im Spreadsheet von Excel eine Tabelle, die zunächst alle Felder der View enthält. Jedoch sollte man die hinterlegte Query noch anpassen:

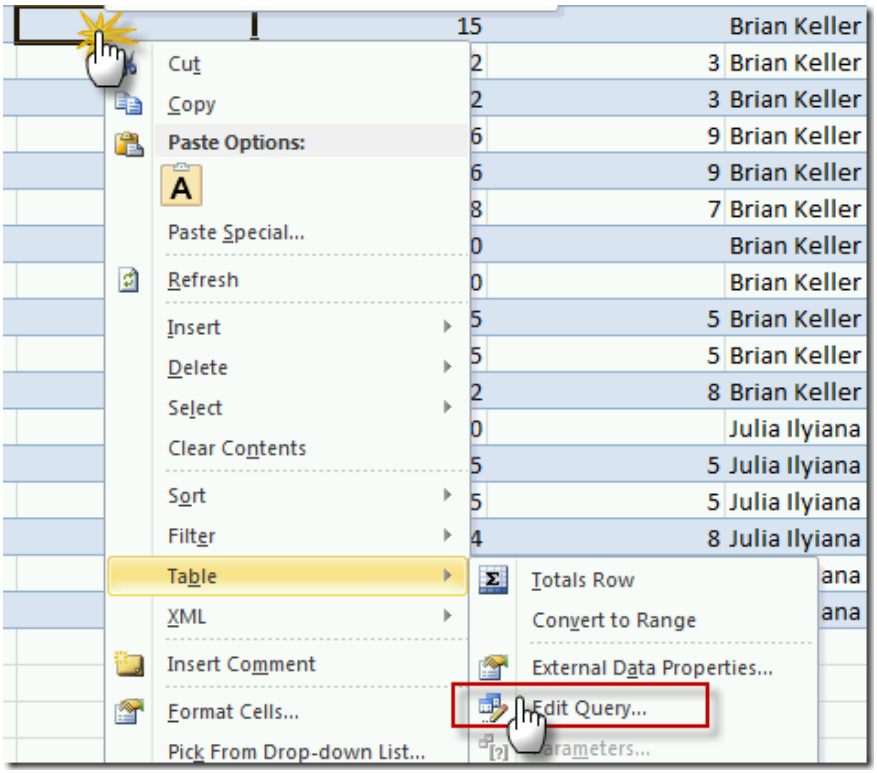



*Abbildung: Ändern der hinterlegten Abfrage*

**Die zu hinterlegende Abfrage lautet:**

```
SELECT
[System_Id] As [Id],[System_Rev] As [Rev]
,[System_ChangedBy] As [Changed By],[System_ChangedDate] As [Changed Date]
,[Microsoft_VSTS_Scheduling_OriginalEstimate] AS [Original Estimate]
,[Microsoft_VSTS_Scheduling_RemainingWork] AS [Remaining Work]
,[Microsoft_VSTS_Scheduling_CompletedWork] AS [Completed Work]
,[System_Title] As [Title],[System_AssignedTo] As [Assigned To]
,[ProjectNodeName] AS [Team Project]
,[AreaName],[AreaPath],[IterationName],[IterationPath]
FROM
[Tfs_Warehouse].[dbo].[WorkItemHistoryView]
WHERE
/** Filter team project **/
[ProjectNodeName] LIKE 'XXX'
```

/** Filter correction entries **/

AND [RevisionCount] IS NOT NULL

/** Filter empty entries **/

AND ([Microsoft_VSTS_Scheduling_OriginalEstimate] IS NOT NULL

OR [Microsoft_VSTS_Scheduling_RemainingWork] IS NOT NULL

OR [Microsoft_VSTS_Scheduling_CompletedWork] IS NOT NULL)

Die Tabelle enthält jedoch immer noch die stetig wachsenden bzw. sinkenden Summen von Completed bzw. Remaining Work und nicht die am jeweiligen Tag von einem Benutzer eingetragene Zeit. Um dies zu erreichen wird eine berechnete Spalte mit folgender Formel zur Tabelle hinzugefügt:

=[@[Completed Work]]-IF(AND(ISNUMBER(INDIRECT("J" & ROW([@Title])-1));INDIRECT("E" & ROW([@Title])-1)=[@Id]);INDIRECT("J" & ROW([@Title])-1);0)

O18 =[@[Completed Work]]-IF(AND(ISNUMBER(INDIRECT("J" & ROW([@Title])-1));INDIRECT("E" & ROW([@Title])-1)=[@Id]);INDIRECT("J" & ROW([@Title])-1);0)

| | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Id | Rev | Changed By | Original Estimate | Remaining Work | Completed Work | Assigned To | Team Project | Changed Date | Title | Completed Diff |
| 2 | 2 | 2 | Julia Ilyiana | | 15 | | Brian Keller | Demo.Scrum | 11.03.2013 22:43 | Test Task 2 | 0 |
| 3 | 2 | 3 | Julia Ilyiana | | 12 | 3 | Brian Keller | Demo.Scrum | 11.03.2013 22:43 | Test Task 2 | 3 |
| 4 | 2 | 4 | Julia Ilyiana | | 12 | 3 | Brian Keller | Demo.Scrum | 11.03.2013 22:43 | Test Task 2 | 0 |
| 5 | 2 | 5 | Julia Ilyiana | | 6 | 9 | Brian Keller | Demo.Scrum | 11.03.2013 22:44 | Test Task 2 | 6 |
| 6 | 2 | 6 | Julia Ilyiana | | 6 | 9 | Brian Keller | Demo.Scrum | 11.03.2013 22:44 | Test Task 2 | 0 |
| 7 | 2 | 7 | Julia Ilyiana | | 8 | 7 | Brian Keller | Demo.Scrum | 11.03.2013 22:44 | Test Task 2 | -2 |
| 8 | 3 | 2 | Julia Ilyiana | | 20 | | Brian Keller | Demo.Scrum | 11.03.2013 22:43 | Test Task 3 | 0 |
| 9 | 3 | 3 | Julia Ilyiana | | 20 | | Brian Keller | Demo.Scrum | 11.03.2013 22:43 | Test Task 3 | 0 |
| 10 | 3 | 4 | Julia Ilyiana | | 15 | 5 | Brian Keller | Demo.Scrum | 11.03.2013 22:44 | Test Task 3 | 5 |
| 11 | 3 | 5 | Julia Ilyiana | | 15 | 5 | Brian Keller | Demo.Scrum | 11.03.2013 22:44 | Test Task 3 | 0 |
| 12 | 3 | 6 | Julia Ilyiana | | 12 | 8 | Brian Keller | Demo.Scrum | 11.03.2013 22:44 | Test Task 3 | 3 |
| 13 | 4 | 2 | Julia Ilyiana | 20 | 20 | | Julia Ilyiana | Demo.CMMI | 11.03.2013 23:07 | Test Task | 0 |
| 14 | 4 | 3 | Julia Ilyiana | 20 | 15 | 5 | Julia Ilyiana | Demo.CMMI | 11.03.2013 23:07 | Test Task | 5 |
| 15 | 4 | 4 | Julia Ilyiana | 20 | 15 | 5 | Julia Ilyiana | Demo.CMMI | 11.03.2013 23:08 | Test Task | 0 |
| 16 | 4 | 5 | Julia Ilyiana | 20 | 14 | 8 | Julia Ilyiana | Demo.CMMI | 11.03.2013 23:08 | Test Task | 3 |
| 17 | 4 | 6 | Julia Ilyiana | 20 | 24 | 0 | Julia Ilyiana | Demo.CMMI | 11.03.2013 23:08 | Test Task | -8 |
| 18 | 4 | 7 | Julia Ilyiana | 20 | | 24 | Julia Ilyiana | Demo.CMMI | 12.03.2013 00:21 | Test Task | 24 |

*Abbildung: Die Ergebnistabelle in Excel*

Die Spalte “Completed Diff” enthält dann die Differenz zum vorherigen Eintrag also zur letzten Work Item-Änderung.

Einziges Manko: Die eingetragenen Zeiten werden immer auf den Tag “gebucht” an dem die Eintragung ins Work Item erfolgte. Ein Benutzer kann also nicht nachträglich für letzte Woche Zeiten eintragen. Diesen Punkt gehen wir im nächsten Blog-Post an in dem wir uns mit Erweiterungen beschäftigen.

Sollten Sie bereits zu diesen TFS-Grundfunktionen Fragen haben, sprechen Sie uns an:

Sven.Hubert@aitgmbh.de

## Verwandte Artikel

- Agiles Arbeiten im Team
- Visual Studio und TFS 2012 Update 2 verfügbar
- Artikel: dotnetpro 04/2013 – Nicht nur mobil

Benötigen Sie Unterstützung bei der Software-Entwicklung und Architektur von .NET basierten Lösungen oder bei Einführung und Anpassung von Visual Studio / Microsoft Test Manager / Team Foundation Server?

Wir stehen Ihnen unter info@aitgmbh.de gerne zur Verfügung.

# Kapitel 2: Testing

## Matthias Zieger (Microsoft) – Effiziente Qualitätssicherung und Testen mit Visual Studio 2012

### Arbeitgeber

• Microsoft Deutschland GmbH

### Technologieschwerpunkte

• Application Lifecycle Management

• Visual Studio

• Team Foundation Server

• SW Qualität

### Mein Urlaubstipp

Meine Lieblingsorte sind die Nationalparks im Südwesten der USA, also z.B. Bryce Canyon, Arches National Park, Canyonlands, Am faszinierendsten sind die Weite der Landschaft und die unterschiedlichen Gesteinsformationen, die einen Einblick in die Erdgeschichte erlauben.

### Meine Inseltechnologie

Am meisten beschäftigt natürlich das Thema ALM, weil es jeden Kunden betrifft, der professionell Software entwickelt. Am meisten beeindruckt mich immer wieder neue Hardware. Rechenleistung, die vor 10 Jahren noch ganze Server-Räume füllte, kann man heute mit sich rumtragen…

## Effiziente Qualitätssicherung und Testen mit Visual Studio 2012

Das folgende Whitepaper/Artikel gibt einen Einblick in die Möglichkeiten der Qualitätssicherung mit Visual Studio 2012. Zielgruppe sind Tester, Testmanager und Verantwortliche für Qualitätssicherung in Software-Projekten.

Dabei werden Themen wie Anforderungsmanagement, Testmanagement, Testautomatisierung sowie Testvirtualisierung und deren Umsetzung mit Visual Studio 2012 betrachtet. Am Ende gibt der Autor einen Überblick über weitere Qualitätssicherungsthemen, die weit über das eigentliche Testen hinausgehen, aber zum Methodenbaukasten eines Qualitätsverantwortlichen gehören sollten.

### Überblick und Zielgruppe

Das folgende Whitepaper soll einen Einblick in die Möglichkeiten der Qualitätssicherung mit Visual Studio 2012  geben mit Schwerpunkt auf das funktionale Testen. Betrachtet werden dazu vor allem Visual Studio 2012, Test Professional 2012 und Team Foundation Server 2012. Das Whitepaper richtet sich an Tester, Testmanager und Verantwortliche für Qualitätssicherung in Software-Projekten. Dabei werden Themen wie Anforderungsmanagement, Testmanagement, Testautomatisierung sowie Testvirtualisierung und deren Umsetzung mit Visual Studio 2012 betrachtet. Am Ende geben wir einen Überblick über Qualitätssicherungsthemen, die weit über das eigentliche Testen hinausgehen, aber zum Methodenbaukasten eines Qualitätsverantwortlichen dazu gehören sollten.

### Hauptziele von VS 2012 im Testbereich

- Bessere Verbindung von Fachseite, Entwicklung und Qualitätssicherung durch Integration in eine ALM-Plattform
- Bereitstellung neuer, integrierter Werkzeuge speziell für Testmanager, Testanalysten und manuelle Tester
- Unterstützung agiler Testmethoden
- Unterstützung bewährter Technologien wie Windows 7, .NET 4, 64 bit Testumgebungen
- Unterstützung der Trends im Bereich Mobility wie Windows 8
- Unterstützung neuester Webtechnologien wie HTML5
- Höhere Testeffizienz durch Testautomatisierung
- Einfacherer Fehleranalyse durch automatisches Aufzeichnen wichtiger Testdaten
- Einfache Verwaltung physikalischer und virtueller Testlabore in heterogenen Umgebungen wie z.B. Microsoft Hyper-V und VM-Ware

- Mit Visual Studio 2012 Update 2 (VS 2012.2) gibt es die Möglichkeit, das Testmanagement über einen Browser zu bedienen. Dies erspart für einige Szenarien das Rollout des Microsoft Test Managers

## Anforderungsbasiertes Testen

Sehr häufig sind die Themen Anforderungsmanagement und Testmanagement sowie Testausführung voneinander getrennt. Dabei kommt es häufig zu missverstandenen Anforderungen, falschen oder fehlenden Testfällen und damit zu Kostenüberschreitungen und Qualitätsmängeln. Aus diesen Gründen ist es notwendig, die Themen Anforderungsmanagement und Testmanagement enger miteinander zu verbinden. Die Testfälle sind dann direktes Ergebnis der Anforderungen, eine Testausführung und aufgetretene Fehler lassen sich bis zu den Anforderungen zurückverfolgen.

**Anforderungs- und Testmanagement mit Visual Studio 2012**

Visual Studio 2012 bietet eine enge Integration der Themen Testausführung, Testmanagement, Anforderungsmanagement, Test-Auswertung, Fehlerverfolgung, und Buildmanagement. Nach der Erstellung und Überprüfung der funktionalen und nichtfunktionalen Anforderungen kann man Testfälle entwerfen und jeden Testfall den relevanten Anforderungen, Use Cases oder User Stories (je nach Vorgehensmodell) zuordnen. Außerdem ist es möglich, zu jedem manuellen Testfall eine Testautomatisierung zuzuordnen.

Im „Microsoft Test Manager" als Teil von Visual Test Professional bzw. Visual Studio 2012 Ultimate in der Integration mit dem Team Foundation Server haben Sie folgende Funktionen integriert:

- Verschiedene Eingabemöglichkeiten der Anforderungen, u.a. im Testmanager, Webbasiert, Excel, Visual Studio Team Explorer
- Erstellung der Testpläne, Testsuiten und Testschritte
- Testplanung auf Basis der Anforderungen
- Testparametrisierung (datengetriebene Tests)
- Aufbau einer Testfall- und Testschrittbibliothek
- Verlinkung zum Buildmanagement
- Steuerung für manuelle und automatisierte Testausführung
- Aufzeichnen relevanter Daten während des Testens
- Testauswertung, Fehlerverfolgung, Reporting

**Unterstützung agiler Testmethoden**

In den letzten Jahren haben agile Entwicklungsmethoden wie Scrum massiv an Bedeutung gewonnen. Die sollte auch im Testbereich seine Entsprechung finden, d.h. auch hier müssen agile Methoden an Bedeutung gewinnen. Eine Möglichkeit ist, hier das sogenannte **Explorative Testen** zu nutzen.

Exploratives Testen bedeutet, eine Anwendung zu testen, ohne vorher genau einen definierten Satz an Testfällen und Testschritten zu erstellen.

Der Microsoft Testmanager hilft dabei, indem die Aktionen, die während des explorativen Tests ausgeführt werden, aufgezeichnet werden können. Man kann Bildschirmabbildungen, Kommentare, Dateianhänge sowie Audio- und Videokommentare aufzeichnen. Diese Aufzeichnung ermöglicht es, jeden Fehler, der gefunden wurde, automatisch zu protokollieren und zu speichern, sowie daraus Testfälle zu generieren, die dann in eine Regressions-Testsuite einfließen können. Ausgangspunkt für einen explorativen Test ist immer eine Nutzeranforderung bzw. ein Szenario, die als WorkItem im TFS hinterlegt sind.

Wenn Fehler oder Testfälle erstellt werden, werden diese automatisch mit der zugrunden liegenden Anforderung verknüpft, d.h. es gibt eine vollständige Unterstützung formaler Anforderungen trotz agiler Vorgehensweise und ist somit beispielsweise CMMI konform bzgl. Anforderungen wie Auditierbarkeit und Traceability. Die Arbeitsaufgabe und alle Testfälle, die während der explorativen Testphase erstellt werden, werden automatisch dem Testplan hinzugefügt.

**Aufzeichnen relevanter Daten während des Testens in der Testumgebung**

Während einer Testausführung müssen normalerweise sehr viele Daten erfasst werden. Hauptsächlich sind das Daten der Konfiguration der Testumgebung, Prozessdaten der zu testenden Anwendung und Code-Testabdeckungsinformationen.

Im Detail können in der Testumgebung folgende Daten erfasst werden:

**„Action log“ und „Action Recording“**

- Aufzeichnen der Klick-Pfade bei manueller Testausführung
- Erlaubt Erstellung automatischer Test-Protokolle bei manuellen Tests
- Testschritte können ohne Benutzerinteraktion beschleunigt wiedergegeben werden (Fast Forward Replay)
- Ist eine mögliche Grundlage für die Testautomatisierung

**Videoaufzeichnung und Screenshot während eines Testlaufs**

- Zeichnet ein Video der Testsituation auf
- Im Fehlerfall kann zusätzlich automatisch oder manuell ein Screenshot der Fehlersituation erzeugt werden
- Für manuelle und automatisierte Tests
- Hilft, Testabläufe besser zu verstehen
- Gut geeignet auch für Usability Testszenarien, da benötigte Zeiten bei manuellen Tests mit protokolliert werden

**IntelliTrace**

- Sammelt Informationen, die helfen, die Fehlersuche und -diagnose zu vereinfachen
- Verbessert massiv die Produktivität beim Debugging
- Aus den IntelliTrace-Daten kann die Test-Session auf einem Entwickler-Rechner wiederhergestellt werden und vereinfacht somit, Fehler schneller zu reproduzieren

**Test Impact Analyse**

- Sammelt Informationen, welche Methoden der getesteten Anwendung aufgerufen wurden während eines Testlaufs
- Ermittelt zusammen mit der Code-Änderungsrate aus der Versionierung und dem Build-System, welche Testfälle von Code-Änderungen betroffen sind
- Hilft dabei, Testfälle zu priorisieren und zu selektieren, insbesondere bei Regressionstestszenarien

**ASP.Net Client Proxy für IntelliTrace und Test Impact Analyse**

- Sammelt Informationen über http-Aufrufe von einem Client zu einem Web Server
- Kann genutzt werden für IntelliTrace und Test Impace Analyse von Web-Anwendungen

**ASP.NET Profiler**

- Sammelt Performance-Daten (Profiling) bei ASP.NET Web Anwendungen

**Code Coverage**

- Ermittelt selbständig, wie hoch die Code-Abdeckung während eines Testlaufs ist
- Hilft, die Qualität der Testfälle in Bezug zur Code-Basis einzuschätzen und zu verbessern

**Event Log**

- Zeichnet automatisch Ereignisse aus dem Event-Log (Ereignisanzeige) auf
- Fügt die Event-Log Informationen dem Testergebnis hinzu

**Netzwerk Emulation**

- Während des Tests kann künstlich die Netzwerkbandbreite eingeschränkt werden
- Hilft dabei, realistischere Testszenarien aufzubauen

**System Informationen**

- Sammelt automatisch Systeminformationen über die eingesetzte Testumgebung, u.a. Betriebssystemversion, Patch-Level, eingestelltes Gebietsschema, Bildschirmauflösung und vieles mehr

**Eigene Diagnose-Daten**

Es ist möglich, einen eigenen Diagnose Adapter zu bauen, der relevante Daten der eigenen Anwendung während des Testausführung sammelt. Das können z.B. eigene Events, Log-Files oder Datenbankinhalte sein. Diese können an die Testausführung angehängt werden und stehen dann dem Testanalysten bzw. Entwickler zur Fehleranalyse zur Verfügung.

**Verbindung zur Testauswertung**

Alle gesammelten Informationen werden automatisch an die Testergebnisse angehängt. Über die Verbindung der Testfallverfolgung und Testauswertung mit dem Bugtracking bzw. Fehlermanagement kann man mit Hilfe dieser Informationen sehr schnell den Fehler nachstellen und beheben. Damit können die „Develop-Test-Repro-Fix" Zyklen massiv verkürzt werden.

## Testautomatisierung

Die Testautomatisierung hat die Aufgabe, die Testeffizienz zu verbessern sowie für verlässlichere Testergebnisse zu sorgen. Insbesondere spielt Testautomatisierung eine große Rolle bei Regressionstest und bei agilen Praktiken wie z.B. „Continues Integration and Test".

**Unterstütze Technologien**

In Visual Studio 2012 können verschiedene Typen von automatisierten Tests erstellt werden. Dabei können unterschiedliche GUI/Oberflächentechnologien getestet werden, wie z.B. Windows Forms, WPF und Webanwendungen. Neben Web-Anwendungen werden Oberflächentechnologien unterstützt, die entweder MS AA (Active Accessability) oder MS UIA (User Interface Automation) unterstützen.

**Unterstützte Testarten**

Die folgende Tabelle zeigt die Automatisierungstypen, die als Teil eines Testplans aus dem Test Manager bzw. aus Visual Studio heraus gestartet werden können:

| Testtyp | Details | Ausführung aus Visual Studio | Ausführung innerhalb eines Testplans im Test Manager über Verbindung zu einem logischen Testfall |
|---|---|---|---|
| Coded UI Tests | Tested die grafische Oberfläche einer Anwendung durch Simulation von Benutzerinteraktionen | Ja (ab VS Premium) | Ja |
| Unit Tests | Test von Quellcode auf der Methodenebene.<br>Eine Neuerung in Visual Studio 2012 ist, das auch andere Testframeworks außer MS Test eingebunden werden können, z.B. | Ja (ab VS Prof.) | Ja |
| Database Unit Tests | Test einer Stored procedure, DB Function oder eines Triggers in einer Databank. | Ja (ab VS Premium) | Ja |
| Load tests | Test der Anwendungsperformance und Stresstests unter Nutzung vorhandener Unit-Tests, Web Performance Tests oder Coded UI Tests | Ja (VS Ultimate)<br>Anm: VS Ultimate enthält unlimitierte vrituelle User,d.h. es kann unbegrenzt Last zu Verfügung gestellt werden. | Ja |
| Web Performance Tests | Testen von Web-Anwendungen auf Protokollebene durch Absendung von http Requests und validieren der http Responses | Ja (VS Ultimate) | Ja. |
| Generic Tests | Testen über API-Aufrufe oder Kommandozeilenfunktionen der zu testenden Anwendungen | Ja (ab VS Premium) | Ja |

**Oberflächen-Test-Automatisierung mit VS 2012 in der Praxis**

VS 2012 (ab der Premium Variante) bietet die Möglichkeit, „Coded UI Tests" zu erstellen. Diese Tests sind Testautomatisierungs-Programme bzw. Testtreiber. Als Technologie für diese Testtreiber wird das .NET 4 .5 Framework genutzt, als Testprogrammiersprachen stehen C# und VB.NET zur Verfügung. Diese „Coded UI Tests" führen Aktionen auf der Benutzerschnittstelle aus und validieren, also überprüfen, die Korrektheit der Darstellung der vorhandenen Oberflächenelemente und deren Inhalte.

Das bietet folgende Vorteile:

- Kein zusätzliches Know-How für eine neue Testsprache notwendig
- Nutzung sämtlicher Komfort-Features des Visual Studio 2012 für die Testautomatisierung
- Nutzen aller Möglichkeiten des .NET 4.5 Frameworks für die Testautomatisierung
- Integration mit der Team Foundation Server Infrastruktur (u.a. für Validation Builds, Versionierung der Testtreiber, Fehlermanagement und Test-Reporting)

In VS 2012 gibt es drei Möglichkeiten, Testautomatisierungsscripte zu erstellen:

- Testfall-Recording
- Nutzung des Coded UI Test Builders
- Umwandlung der beim manuellen Test aufgezeichneten Action Recordings eines manuellen Test

**Testfall-Recording**

- Zeichnet alle Benutzerinteraktionen auf der Programmoberfläche (GUI) auf
- Erzeugt Testscripte in C# oder VB aus diesen Benutzerinteraktionen
- Validierungen für unterschiedlichste Merkmale der Oberflächenelemente lassen sich hinzufügen

**Testautomatisierung mit dem „Coded UI Test Builder“**

- Erlaubt sehr schnell, neue GUI-Elemente und Verifizierungen für einen neuen oder zu einem bestehenden automatischen Testfall hinzuzufügen
- Einfache Nutzung
- Erzeugt Code sowohl für die Ansteuerung der GUI-Elemente als auch Validierung

**Umwandlung eines manuellen Tests in einen automatisierten Test**

- Erzeugt automatisch Testscripte in C# oder VB aus den Action Recordings der manuellen Testausführung
- Tests können dann wahlweise manuell oder automatisiert ausgeführt werden
- Verkürzt die Zeit, um automatisierte Testscripte zu erzeugen

**Datengetriebene automatisiert Tests**

Nachdem die logischen Testfälle erstellt sind, sollen diese natürlich mit verschiedenen Testdaten ausgeführt werden. Dazu bietet Visual Studio 2012 die Möglichkeit, eine Datenquelle zum coded UI Test hinzuzufügen und diesen dadurch zu einem Datengetriebenen Test umzuwandeln. Jede Zeile in der Datenquelle ist dann eine Iteration oder Ausprägung des logischen Testfalls. Das Testergebnis setzt sich dann aus den Einzelergebnissen jeder Testiteration zusammen.

## Last- und Stresstests

Das Ziel von Load Test ist es, zahlreiche simultane Benutzerzugriffe auf eine Server-Applikation zum gleichen Zeitpunkt zu simulieren. Damit soll sichergestellt werden, dass die Korrektheit der Anwendung auch bei starker Last gegeben ist.
Visual Studio 2012 Ultimate unterstützt dabei drei Modelle, Last zu generieren:

**Loadtest für Webanwendungen**

- Ermöglicht den Vergleich von zwei Testläufen
- Simuliert viele gleichzeitige Benutzer, die HTTP Requests ausführen gegen einen beliebigen Web-Server
- Zahlreiche Parameter wie z.B. Browser, Bandbreite, Think-Time usw. lassen sich simulieren
- Testscripte lassen sich in C# oder VB konvertieren und damit die Mächtigkeit des .NET Frameworks für Lasttests nutzen.

**Wiederverwendung von Unittests für Loadtestszenarien**

- Ist ebenfalls eine etablierte Funktion der Visual Studio Team System 2008 Test Edition
- Ermöglicht Loadtestszenarien für Komponenten, die nicht Webbasierend sind, z.B. Datenzugriffskomponenten, Workflowkomponenten oder sonstige Kommunikationskomponenten
- Wiederverwendungseffekte führen zu Kosteneinsparungen

**Wiederverwendung von bereits automatisierten Tests im Lasttets**

- Automatisierte GUI-Tests lassen sich in Loadtestszenarien wiederverwenden
- Dadurch sind eigene Anwendungen als Testtreiber für Loadtestszenarien nutzbar
- Testvirtualisierung mit Visual Studio  Lab Management

## Virtualisierung von Testumgebungen

Microsoft Visual Studio Lab Management ist eine Erweiterung des Microsoft Test Managers zum Verwalten und Benutzen von virtuellen Maschinen zum Build, Deployment und Testen von Anwendungssystemen. Visual Studio Lab Management ist integriert mit dem System Center Virtual Machine Manager (SCVMM) zur Verwaltung mehrerer pysikalischer Computer, welche als Gast für die virtuellen Maschinen agieren.

Visual Studio Lab Management ermöglicht ein schnelles Deployment virtueller Testumgebungen zur Ausführung von Tests und Automatisierung von Builds. Damit kann man sehr schnell komplexe Konfigurationen erstellen, die weitestgehend der Produktivumgebung entsprechen. Die

Testvirtualisierung ermöglicht durch die Nutzung der Snapshot-Technologien eine schnellere Reproduzierbarkeit von Fehlersituationen. Ausserdem sind Testumgebungen schneller in definierte Zustände zurücksetzbar. Zusammen mit dem Testmanagement verringert sich massiv der Aufwand für professionelles Testen innerhalb des Application Lifecycle Managements.

**Testvirtualisierungspraktiken mit Visual Studio  Lab Management**

- Rücksetzen von Umgebungen - stellt sicher, dass die Testumgebungen immer in einem definierten Zustand sind
- Snapshots von Testumgebungen – ermöglicht jederzeit, in verschiedene Testumgebungszustände zu wechseln
- In Fehlersituationen können Snapshots automatisch erzeugt werden und an Testergebnisse angehängt werden
- Klonen von Testumgebungen – ermöglicht durch die Netzwerkisolierung die Erstellung von Kopien komplexer Testumgebungen, die aus mehreren Virtuellen Maschinen bestehen können
- Integration mit Team Foundation Server und Buildmanagement – ermöglicht einen durchgehenden Workflow vom zentralen Build über automatisches Deployment bis hin zur Testautomatisierung
- Verwalten von Umgebungsbibliotheken – bietet die Möglichkeit, das Deployment und die Tests auf korrekten, definierten Umgebungen aus einer VM-Bibliothek durchzuführen

**Unterstützung von Standard-Testumgebungen**

Neben den Hyper-V basierenden Testumgebungen auf Basis des System Center Virtual Machine Managers (SCVMM) werden mit Visual Studio 2012 auch sogenannte Standardumgebungen unterstützt.
Standardumgebungen können sowohl virtuelle Maschinen auf Basis beliebiger Virtualisierungstechniken (wie z.B. VMWare) sein als auch physische Computer sein.

## Fehlerverfolgung, Auswertung und Reporting

Visual Studio  im Zusammenspiel mit dem Team Foundation Server kann dabei unterstützen, die Qualität im Softwareentwicklungsprojekt besser einzuschätzen. Insbesondere gibt es zahlreiche Auswertungsmöglichkeiten.

- Welche SW-Builds haben Code-Änderungen im Zusammenhang mit Bug Fixes, Change Requests, Anforderungen usw.
- Welche Testfälle müssen im Regressionstest basierend auf Codeänderungen erneut in einem Testplan zusammengestellt werden?
- Sind die Testfälle fertig definiert, und wenn ja, wie ist der Testfortschritt?

Es ist möglich, beliebige eigene Reports zu definieren, die verschiedenste Aspekte des Application Lifecycle Managements umfassen können

- Anforderungsmanagement
- Changemanagement
- Versionierung
- Build- und Releasemanagement
- Testmanagement- und Testausführung
- Fehlerverfolung

## Entwicklung und Test stärker verbinden

Entwickler können die im Test gefundenen Fehler viel besser als bisher nachvollziehen, indem die Testumgebung als Snapshot zum Zeitpunkt des Auftretens des Fehlers vorliegt.

Die neuen Features im Testmanagement, der Datenaufzeichnung während der Testausführung und die Testvirtualisierung sorgen dafür, dass Fehler schneller analysiert und behoben werden können. Fehlermeldungen mit dem Status „ Nicht Reproduzierbar“ sollten damit deutlich weniger werden.

Dadurch sind Entwickler und Tester besser als jemals zuvor in der Lage, miteinander statt gegeneinander zu arbeiten und damit die Qualität der Software zu verbessern.

## Zusammenfassung: Qualität ist mehr als nur Testen

Typischerweise manifestiert sich die Qualität eines komplexen Softwaresystems in den während der Erstellung gelieferten und produzierten Artefakte. Dazu zählen insbesondere:

- Die Anforderungen als Basis für alle anderen Artefakte
- Abstrakte Prozessmodelle oder konkrete Implementierungsmodelle (UML, DSL, Datenbankmodelle) und deren Diagramme
- Quellcode, Scripte, Konfigurationselemente, Handbücher
- Buildscripte
- Testpläne, Testfälle, Testumgebungen und Testdaten

Alle diese Artefakte bzw. Arbeitsprodukte müssen qualitätsgesichert werden. In welchem Umfang dies geschieht, ist abhängig von den Rahmenbedingen des Projektes, beispielsweise Compliance-Anforderungen (SOX, Basel II) oder Regularien (z.B. Medizintechnik ISO 13485, IEC 62304 und diverse FDA-Regeln).

Visual Studio 2012 bietet dabei eine Unterstützung der Qualtitätssicherung neben dem reinen Testen über den gesamten Application Lifecycle an (u.a. Anforderungsmanagement, UML-Modellierung, DSL-Toolkit, Architekturvalidierung- und Visualisierung, statische und dynamische Code-Analyse, Code Coverage Analyse, Check-In Regeln, Gated Check-Ins um nur einige zu nennen).

Damit ist klar, dass Qualität nicht in ein System hineingetestet werden kann. Vielmehr ist es Aufgabe des Testens, letztendlich Qualität zu bestätigen oder zu widerlegen – und zwar möglichst früh im Anwendungsentwicklungszyklus. Eine Abwesenheit von Fehlern im Bugtracking-Werkzeug lässt meist auf eine unzureichende Testplanung oder falsche Testfälle bzw. Testdaten schliessen, Ausnahmen bestätigen die Regel.

VS 2012 mit den dazugehörigen Komponenten „Microsoft Test Manager", „Test „Lab Management" sowie „Team Foundation Server" unterstüzen dabei, hochwertige Testpläne mit Testfällen auf Basis von validen Anforderungen zu erstellen und diese Testfälle sehr effizient zu automatisieren.

Durch die hohe Integration in das Visual Studio und den Team Foundation Server und die Möglichkeiten zur Fehleranalyse bzw. Reproduktion ist Visual Studio 2012 nicht nur  das Tool der Wahl für Softwareentwickler, sondern auch für Tester, Testmanager und Testanalysten.

# Kapitel 3: Coding

## Boris Wehrle (AIT GmbH) – Alltagstaugliche Programmierrichtlinien: Spickzettel für Entwickler

### Arbeitgeber

• AIT GmbH & Co. KG

### Technologieschwerpunkte

• Software Architekt für Industrie- und B2B-Applikationen
• .NET UI Technologien (WPF, WinForms, ASP.NET MVC)
• Web Technologien
• Windows Azure
• Projektabwicklung, Trainings und Coaching

### Mein Urlaubstipp

Peru – alle Klimazonen konzentriert in einem Land. Wer gern einmal alle Klimazonen der Erde innerhalb von drei Wochen bereisen möchte, ist in Peru genau richtig. Von staubtrockenen Wüsten über tropischen Dschungel bis hin zu 7.000er Bergen gibt es hier alles. Beim Hinabklettern in den tiefsten Canyon der Welt oder beim Trekken mit Pferden durch die Berge fernab der Zivilisation kann man perfekt abschalten.

### Meine Inseltechnologie

Entwicklungsrichtlinien werden sehr häufig als Ballast empfunden. Sie fristen als dicker, eingestaubter Stapel Papier ihr Dasein im Regal und werden immer dann herausgezogen, wenn ein neuer Kollege in der Tür steht. Gelebte, von allen Beteiligten akzeptierte Vereinbarungen können die Arbeit ihm Team

durchaus erleichtern. Wie müsste ein solcher Regelsatz aussehen? Reduziert auf das Wesentliche – auf maximal ein Blatt Papier!

## Alltagstaugliche Programmierrichtlinien: Spickzettel für Entwickler

Regeln für das Erstellen von Code bilden die Basis für das Programmieren im Team – aber nur, wenn sie auch  tatsächlich im Alltag gelebt werden und nicht im Regal verstauben.

Programmierrichtlinien sind laut Wikipedia ein Satz von Regeln, die beim Schreiben von Quelltext zugrunde liegen und denen sich ein Programmierer unterwirft. Viele Unternehmen definieren solche Regeln in umfangreichen Dokumenten, die jedes Detail genau festlegen. Aber mal Hand aufs Herz: Halten Sie sich an die Richtlinien Ihres Unternehmens? Wissen Sie genau, was in diesen festgelegt ist? Und wann haben Sie sich diese zum letzten Mal zu Gemüte geführt?

Entwicklungsrichtlinien werden sehr häufig als ungeliebter Ballast empfunden. Ihr Dasein fristen sie dann oft nur als dicker, eingestaubter Stapel Papier in irgendeinem Regal; ihre Stunde schlägt immer dann, wenn ein neuer Kollege in der Tür steht. Solche Vereinbarungen können die Arbeit im Team durchaus erleichtern – wenn sie von allen Beteiligten akzeptiert und auch gelebt werden.

Aber wie müsste ein solcher Regelsatz aussehen? Wie müsste er reduziert sein auf das Wesentliche – nämlich auf einen Umfang von maximal einem Blatt Papier?

Zweck von Programmierrichtlinien Das Ziel von Programmierrichtlinien ist es, die Zusammenarbeit der Mitarbeiter in einem Entwicklungsteam oder in einem Unternehmen zu vereinfachen. Indem die Regeln einen einheitlichen Programmierstil [1] definieren, ermöglichen sie es den Entwicklern, sich im Code eines Kollegen bei Bedarf schnell zurechtzufinden und diesen weiterzuentwickeln. Dies wiederum bietet die Möglichkeit, Entwicklungsteams flexibel zusammenzusetzen, auch unter Einbeziehung externer Dienstleister, und damit schnell auf sich verändernde Anforderungen reagieren zu können.

Die Richtlinien umfassen dabei einfache Regeln wie zum Beispiel zur Quellcodeformatierung und Variablenbenennung, aber auch komplexe Elemente wie Entwurfs- und Architekturmuster.

Regeln zur Quellcodeformatierung zielen dabei auf eine Vereinfachung der Nachvollziehbarkeit von Veränderungen in der Quellcodeverwaltung ab. Eine unterschiedliche Einrückungstiefe bei einzelnen

Entwicklern zum Beispiel lässt eine ganze Datei verändert erscheinen, obwohl sich womöglich nur eine einzige Zeile geändert hat.
Eine einheitliche Verwendung von Architekturmustern wiederum vereinfacht langfristig die Wartbarkeit und eine Wiederverwendung von Komponenten in anderen Projekten.

## Regeln für den Regelsatz

Programmierrichtlinien sollten aber selbst einigen Regeln folgen, um effektiv wirken zu können. Folgende Leitsätze haben sich dabei als bedeutsam erwiesen:

- Verweisen Sie auf schon bestehende, allgemein verfügbare Standards, anstatt sie per Copy-and-Paste in ein eigenes Dokument zu überführen.
- Verwenden Sie zum Beispiel die Coding Conventions für C# oder Visual Basic [2, 3] und die Namenskonventionen von Microsoft [4] – aber nennen Sie diese auch nur als Referenzen, statt sie nochmals selbst als eigenes Dokument wiederzugeben.
- Vermeiden Sie Widersprüche zu schon bestehenden, allgemeingültigen Standards. Dies ist unter anderem wichtig, um später gegebenenfalls externe Entwickler einfacher einbeziehen zu können. Hierzu zählen auch die Standardeinstellungen der Entwicklungsumgebung.
- Reduzieren Sie die Regeln auf das, was schnell zu überprüfen ist.
- Programmierrichtlinien entfalten ihre Wirkung nur dann, wenn sie eingehalten werden. Deshalb müssen sie auch unter starkem Projektdruck gültig bleiben. Erfahrungsgemäß funktioniert dies nur bei Regeln, die sich automatisiert durch Werkzeuge prüfen lassen, welche in die Entwicklungsumgebung oder den Build-Prozess integriert sind.
- Hinterfragen Sie jede Regel.
- Jede in den Entwicklungsrichtlinien definierte Regel muss einen Zweck erfüllen, der für alle Beteiligten nachvollziehbar ist. Nur sinnvolle Regeln werden befolgt.

## Ein Beispiel aus der Praxis

Als Beispiel für einen Regelsatz, der auf ein Blatt Papier passt, können die Programmierrichtlinien des Systemhauses AIT gelten, die in Abbildung 1 zu sehen sind [5]. In diesen sind folgende Punkte festgelegt:

- Die Minimalausstattung eines Entwickler-PCs mit Werkzeugen. Neben den Standard-Tools spielen hier insbesondere zusätzliche Analysewerkzeuge wie zum Beispiel ReSharper [6] und Style-Cop [7] eine Rolle.
- Implizit ergeben sich durch die Verwendung bestimmter Tools weitere Festlegungen. So wird zum Beispiel durch die Auswahl des Mind-Mapping-Tools Freemind [8] definiert, in welchem Format

Mind-Maps ausgetauscht werden; die Auswahl des Plug-ins xUnit [9] für Visual Studio wiederum legt gleichzeitig auch fest, mit welchem Framework bei Tests gearbeitet wird.

- Die genutzten Analysewerkzeuge lassen sich durch Regelsätze anpassen, die in Dateien konfiguriert sind. Die im Unternehmen gültigen Einstellungen für Projektmappen sind in der Versionskontrolle hinterlegt und werden von jedem Projekt referenziert. Dazu gehört auch ein Dokument, das den Aufbau einer Projektmappe allgemeingültig beschreibt.
- Für die Entwicklung gibt es weitere Konkretisierungen, denn die Namenskonventionen von Microsoft [3] lassen beispielsweise offen, wie Felder benannt werden. Außerdem werden Kennzahlen für Metriken und die Codeabdeckung durch Unit-Tests definiert. Ergänzt werden diese Regeln durch Richtlinien zur Dokumentation. Die konsequente Verwendung von Layer- und Klassendiagrammen in jedem Projekt macht es für neue Entwickler einfacher einzusteigen.
- Über die Programmierrichtlinien hinaus gelten für die Projekte weitere Regeln für die Zusammenarbeit im Team. Sie vereinfachen zusätzlich den Wechsel von Entwicklern zwischen unterschiedlichen Projekten.
- Zu alldem kommt noch eine gewisse Pflichtlektüre. Eine gemeinsame Wissensbasis verbessert ein gemeinsames Verständnis für den Aufbau einer Anwendung. Dieser Abschnitt verweist auf Artikel oder Bücher, die für jeden Entwickler die Basis darstellen. Dies vereinfacht die Kommunikation im Team und schafft ein gemeinsames Verständnis.

AIT Coding Guidelines
1.0.00807 - 2012-08-07

## Entwicklungsumgebung

- Office 2010 32bit (incl. Visio, OneNote)
- Visual Studio 2010 Premium (CU 3)
- Productivity Power Tools
- Team Foundation Server Power Tools
- Visual Studio 2012 Premium
- ReSharper 7
  Agent Johnson, Agent Smith
- StyleCop 4.7
- Freemind 0.9
- GhostDoc
- Skype
- xUnit + ReSharper-Plug-In

## Projektmappen- und Projekteinstellungen

- Statische Code-Analyse
  $/AIT/CodingGuidelines/AIT.ruleset
- Stylecop
  $/AIT/CodingGuidelines/StyleCopSettings.StyleCop
- ReSharper
  $/AIT/CodingGuidelines/ResharperSettings.resharper
- Code Analysis Dictionary
  $/AIT/CodingGuidelines/CodeAnalysisDictionary.xml

➢ Mehr Produktivität durch die richtige Solutionstruktur

## Pflichtlektüre

- The Art of Unit Testing
- Gelebte Architekturdokumentation in einem globalen Team
- Code Reviews – Jeder kennt's
- Evolution einer Anwendung
- Distributed Scrum – Verteiltes Arbeiten...

## Weitergehende Informationen

- AIT Developer Guide

## Regeln – Entwicklung

- Pro Projektmappe ein Modeling Project mit mind. einem Layerdiagramm
- Pro Projekt ein Klassendiagramm
- Dokumentation öffentlicher Member im Code - Reduzierung auf das Wesentliche
- Klassen werden „grün" verlassen
- Eine Klasse pro Datei
- Einhaltung Microsoft Coding- und Namenskonventionen
  Ausnahmen:
  - Felder „_xxx"
  - Tests „Xxx_xxx_xxx_xx
  - Eventhandler „Xxxx_Xxx
- 40% Code Abdeckung durch Unit- und Integrationstests in Service- bzw. Logikkomponenten
- Kein Check In ohne Task
- Clean Code Practices KISS, YAGNI, DRY

## Regeln – Projekt

- Weekly Scrum
- Tägliche Aktualisierung von Remaining und Completed Work
- Können Abschätzungen nicht eingehalten werden wird dies spätestens bei 50% Umsetzung kommuniziert.
- Keine Arbeit ohne Task

## Ansprechpartner

- Organisatorisches: DAR
- Architektur: LAR, BOW, TRU, PKU
- Infrastruktur: THD, NIO
- Projektsteuerung: LAR, BOW, TRU
- ALM: SVH

*Abbildung 1*

Als Ergänzung nennen die Richtlinien schließlich noch die Ansprechpartner für unterschiedliche Bereiche, die für Diskussionen sowie Unterstützung zur Verfügung stehen.

## Fazit

Beim Verfassen und/oder auch dem regelmäßigen Überprüfen von Programmierrichtlinien gilt es, sich regelmäßig das eigentliche Ziel in Erinnerung zu rufen – nämlich die Zusammenarbeit im Team zu vereinfachen. Denn wie so oft gilt auch hier: Weniger ist mehr.


[1] Wikipedia, Programmierstil, www.dotnetpro.de/SL1303Guidelines1

[2] MSDN, C# Coding Conventions, www.dotnetpro.de/SL1303Guidelines2

[3] MSDN, Visual Basic Coding Conventions, www.dotnetpro.de/SL1303Guidelines3

[4] MSDN, Guidelines for Names, www.dotnetpro.de/SL1303Guidelines4

[5] AIT Coding Guidelines, www.dotnetpro.de/SL1303Guidelines6

[6] ReSharper, www.dotnetpro.de/SL1303Guidelines6

[7] CodePlex, StyleCop, http://stylecop.codeplex.com

[8] SourceForge, FreeMind, http://freemind.sourceforge.com

[9] CodePlex, xUnit, http://xunit.codeplex.com



Mit freundlicher Unterstützung von:

dotnetpro

www.dotnetpro.de

## Lars Roith (AIT GmbH) – Worauf Sie bei .NET 4.5 und .NET 4.0 achten sollten

### Arbeitgeber

• AIT GmbH & Co. KG

### Technologieschwerpunkte

• .NET Entwicklung und Softwarearchitekturen

• Softwareentwicklungsprozesse für etablierte Teams

### Mein Urlaubstipp

Korsika: Unberührte Natur, Berge und Wasser. Und alles in unmittelbarer Nähe. Dazu eine bewegte Historie, gutes Essen, guter Wein.

### Meine Inseltechnologie

Agiler Formalismus: Auch regulierte und stark formalisierte Entwicklungen können von agilen Praktiken profitieren. Dabei muss der vorgeschriebene Formalismus z.B. für Produktvalidierungen aber nicht auf der Strecke bleiben sondern kann sich in die Agilen Methoden integrieren. Für viele unserer Kunden ist dies eine zwingende Bedingung und es ist erstaunlich, wie leicht und schnell agile Praktiken in diesen Teams adaptiert werden, wenn die Gewissheit besteht, das auch dem notwendigen Formalismus Rechnung getragen wird.

## Worauf Sie bei .NET 4.5 und .NET 4.0 achten sollten

“Das .NET Framework 4.5 geht uns nichts an – wir bauen gegen .NET 4.0!” werden Sie jetzt evtl. denken. Doch der Schein trügt. Sobald auf der Maschine auf der Ihre Applikation ausgeführt wird .NET

4.5 installiert ist, wird auch Ihre Applikation Assemblies aus dem neuen .NET Framework verwenden. Wir zeigen Ihnen worauf Sie achten sollten zur Entwicklungs-, Compile- und Laufzeit…
Über die Neuerungen von .NET 4.5 haben wir bereits in unserer Blogserie berichtet: Neu in .NET 4.5

## Warum ist das für mich relevant?

Was vielen unserer Kunden entgangen ist, ist die Tatsache, dass auch Projekte von den Änderungen betroffen sind, die gar nicht gegen .NET 4.5, sondern gegen .NET 4.0 gebaut werden. Der Schein trügt: **das .NET Framework 4.5 ist ein Inplace-Upgrade welches tw. Assemblies aus dem .NET Framework 4.0 austauscht!**
Das hat tw. große Auswirkungen! Microsoft hat die Details zur Kompatibilität und den Änderungen hier zusammengefasst: Breaking Changes in .NET 4.5

## Was bedeutet das zur Laufzeit?

Zunächst einmal zur **Laufzeit:** einmal gegen .NET 4.0 gebaut, kann Ihre Applikation auf dem Client der diese ausführt auf .NET 4.5 treffen. Damit werden auch von Ihrer Applikation die neuen Assemblies angezogen und verwendet. Dadurch können sich merkwürdige Effekte einstellen oder die Applikation gar nicht mehr funktionieren. Wir haben bereits Probleme im Namespace System.Reflection festgestellt, die eine Applikation völlig zum Erliegen gebracht haben.
Das hat in Konsequenz zur Folge, dass Sie Ihre Applikation auf beiden Systemen (eines ohne und eines mit .NET 4.5) testen müssen. Theoretisch komplett… oder aber Sie bauen Ihre Applikation gleich gegen .NET 4.5 und unterstützen dann nur dieses Szenario. Was allerdings dem Support für Windows XP widerspricht. Oder wie es Microsoft ausdrückt:

⚠ **Important:** Note that the .NET Framework 4.5 is not supported on Windows XP.

Und das bringt uns zum Compile…

## Was bedeutet das zur Compile- und Test-Zeit?

Für die Kompilierung bedeutet das, dass auf Build-Umgebungen mit .NET 4.5 auch gegen die ausgetauschten .NET 4.0 Assemblies gebaut wird – selbst wenn in den Visual Studio Projekteinstellungen 4.0 als Target Framework eingestellt ist. Hinzu kommt, dass bei der Verwendung von Team Foundation Server 2012 die Build-Umgebung immer auch .NET 4.5 gleich mitbringt, da der Team Foundation Build Service dieses implizit mitinstalliert. Soweit so gut, das mag ja noch verkraftbar sein – sollte in jedem Falle aber z.B. vor einem Upgrade von TFS 2010 geprüft werden.

Gravierender ist die Sache, wenn man im Build automatisierte Unit-Tests ausführt. Da diese Laufzeitaspekte prüfen, wird im Falle der Ausführung im TFS-Build-Prozess quasi gegen .NET 4.5 getestet. Und daran führt auf dem Build-Server kein Weg vorbei. Einziger Ausweg, um auch auf reinen .NET 4.0 Systemen zu testen, ist es einen Visual Studio 2010 Test Agent auf einer separaten Maschine ohne .NET 4.5 zu installieren und die Tests im Build-Prozess Remote auf dieser Maschine auszuführen. Der Test Agent 2010 ist kompatibel zum neuen TFS 2012, weshalb sich die Ergebnisse auch wieder in den TFS zurücksenden lassen. Wie dies genau konfiguriert werden muss, haben wir bereits in einem früheren Blogbeitrag für Sie zusammengestellt: CodedUI-Tests ohne Lab Management ausführen

## Was bedeutet das zur Entwicklungszeit?

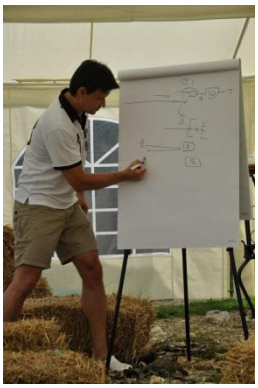
Für die Entwicklung hat das auch Auswirkungen. Bei der Installation von Visual Studio 2012 wird ebenfalls .NET 4.5 implizit installiert. Das hat zur Folge, dass evtl. Entwicklungs-Tools, die in Visual Studio 2010 bzw. mit .NET 4.0 laufen, nicht mehr funktionieren. Uns ist das mit Moles – dem bisherigen Mocking-Framework von Microsoft Research – aufgefallen. Dieses verweigerte seinen Dienst, nach der Installation von VS 2012 und ließ sich auch nach einer Deinstallation von VS 2012 im alten Visual Studio 2010 nicht mehr zum Laufen bringen.

## Fazit

Die Entscheidung, ob man .NET 4.5 unterstützt oder nicht wird einem also durch das Inplace-Upgrade abgenommen. Man kommt nicht umhin Applikationen entsprechend zu testen bzw. den Komplettumstieg zu machen.

# Kapitel 4: Web Development

## Damir Dobric (DAENET GmbH) – Web vNext mit SignalR



### Arbeitgeber

• DAENET GmbH

### Technologieschwerpunkte

• Microsoft VTSP Windows Azure

• Integration MVP

• Windows Azure, WCF, WF, WebApi, SignalR, ServiceBus, Workflow Manager, AppFabric, BizTalk

### Mein Urlaubstipp

Ich habe nicht wirklich einen Lieblings-Urlaubsort. Oder besser ausgedrückt, ich war noch nie in einem Nicht-Lieblings-Urlaubsort :-)

Aber generell bevorzuge ich zwei Typen von Urlaubssituation:

Platz I: Sonne und Strand mit Internet Verbindung

Platz II: Schifahren über 2000m mit Internet Verbindung

### Meine Inseltechnologie

Mich bewegen überwiegend die Aufgaben die große Herausforderung darstellen. Aus diesem Grund ist liegt Schwerpunkt  meiner Expertise in Verteilten Systemen und Integration unterschiedlichsten Anwendungen und Technologien. Deshalb beschäftige ich mich mit Technologien, die die Basis für Solide Architektur darstellen. Meine Lösungen finde ich entlang Technologien wie WCF, WF, WebApi und SignalR und kommen in folgenden Produkten vor: Windows8 mit WinRT, Windows Server, Service Bus,  SharePoint Workflow Manager, BizTalk Server und Windows Azure.

Momentan fokussiere ich „Mainstream“ unter den Namen „Devices & Services“. Aus diesem Grund finde ich das Thema SignalR als eine von Technologien die unsere Denkweise über Web im positiven Sinne verändern könnte.

## Web vNext mit SignalR

Die rasante Entwicklung von neuen Technologien gibt einem Softwareentwickler selbst im Urlaub kaum Zeit zur Erholung. Es wäre bestimmt einfacher und sicher erholsamer, einfach den Sommer zu genießen, als schon wieder irgendeine neue Technologie kennen zu lernen.  So gesehen ist dieser Artikel eine schlechte Nachricht.
Wenn Sie aber in die Welt von neuartigen Anwendungen einsteigen möchten, haben Sie sowieso keine Wahl und sind hier genau richtig. Denn in diesem Artikel geht es darum, eine neue und auf GitHub sehr populäre Open-Source Bibliothek namens SignalR vorzustellen. Wenn Sie Anwendungen entwickeln, oder entwickeln möchten, die als „Connected Distributed Web Systems“ bezeichnet werden, ist SignalR eine valide Option.

### In welcher Zeit leben wir?

Das Web scheint mit einer immer größeren Anzahl an mobilen Geräten am Markt nicht an Bedeutung zu verlieren. Die Apps können nicht unbedingt viel, aber kommen gut an. Man könnte fast behaupten, die Apps könnten die klassischen Webanwendungen ein wenig verdrängen. Es kling fast paradox, aber es steht fest: Das Web war nie wichtiger. Es ist nicht die Absicht dieses Artikels, die Trends bezüglich Apps, Desktop-, oder Webanwendungen zu beleuchten. Vielmehr versuchen wir uns davon zu befreien und treffen eine mutige, aber sinnvolle Annahme: „Das Web ist der Bus“. Nun, zumindest in diesem Artikel, betrachten wir das Internet als Software-Bus-System. Dies gibt uns die Möglichkeit, einigen aktuellen Herausforderungen in der Anwendungskommunikation mit einem neuen Lösungsansatz zu begegnen.

Grundsätzlich kann eine Anwendung (Partizipant im System) eine private oder eine öffentliche IP-Adresse besitzen. Unter Partizipant verstehe ich eine beliebige Anwendung, die in einem Kommunikationssystem interagiert. Das könnte zum Beispiel ein Browser, eine App oder ein Service sein.

In der Regel muss man aber davon ausgehen, dass alle „Partizipanten“ eine private, nicht im Internet veröffentlichte Adresse besitzen und dadurch in der Praxis von außen nicht erreichbar sind. Oder noch ungünstiger: Sie sind hinter einer Firewall „versteckt“ und damit ebenfalls nicht zugänglich.

## Was ist die Anforderung?

Unser Ziel in diesem Artikel ist, solche Partizipanten vom Server aus ansprechen zu können, auch dann, wenn sie nicht direkt erreichbar sind.
Diese Anforderung ist gemeinhin unter dem Namen „Push Notification“ bekannt und stellt eine große Herausforderung dar. Es gibt zahlreichen Szenarien, die häufig entweder gar nicht oder nur schwierig umgesetzt werden können, wenn es darum geht, die Anwendungen in ihrer „geschützten“ Umgebung zu erreichen.

Vor etwa 15 Jahren, als C++ meine Welt dominierte, hätte ich persönlich die Kommunikation über WebSocket [1] mit C++ realisiert. Das wäre auch heute noch möglich, aber leider bedeutet die Vielfalt der verschieden Plattformen, dass WebSocket nicht überall verfügbar ist. Wenn Sie bereits Windows Server 2012 im Einsatz haben und Internet Explorer 10 oder Google Chrome unter Ihren Nutzern bereitstellen können, könnten Sie diese Technologie in Betracht ziehen.

Ich kenne leider niemanden, der diese Anforderungen erfüllen kann. Im Moment ist technologisch so viel im Umbruch, dass meiner Meinung nach sehr schwer ist, die richtigen Entscheidungen zu treffen, um auch nur die nächste halbe Dekade überbrücken zu können.

Zumindest für das Problem der Push Notifications könnte SignalR helfen und eine Lösung für die Zukunft sein.

## Was ist SignalR?

SignalR ist eine Bibliothekensammlung, die Real-Time-Messaging zwischen Partizipanten ermöglicht und dabei verschiedene Protokolle mit einer WebSocket-ähnlichen API unterstützt.

Es handelt sich hierbei um ein ursprünglich privates Projekt der beiden Microsoft-Mitarbeiter Damian Edwards und David Fowler. Das Projekt ist auf GitHub unter [2] zu finden.

Bevor wir uns den Protokollen widmen, betrachten wir, wie eine Push-Lösung allgemein realisiert werden kann.

In Abb. 1 sind die verschiedenen Szenarien dargestellt. Das erste Szenario zeigt eine gewöhnliche Web-Anwendung. Ein Browser schickt einen Request zum Server, aber der Server kann nur eine Response-Message zurückschicken. Wie erwartet, ist ein Request von Server zum Browser nicht möglich. Dieses Szenario ist unsere Ausgangsituation.

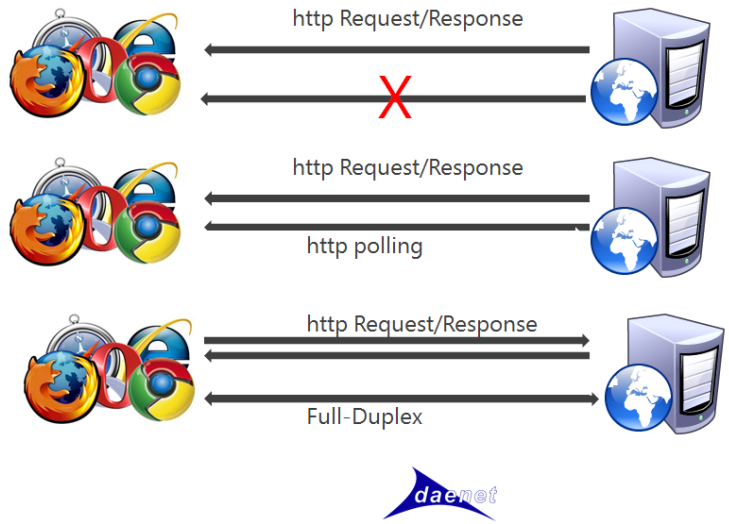



*Abb1-WebInfrastructure*

Im zweiten Szenario realisiert der Browser das sogenannte Polling-Verfahren. Das heißt, der Browser schickt in regelmäßigen Intervallen einen Request zum Server, um ein Ergebnis abzufragen. Das ist eine sehr einfache Realisierung, entspricht aber keinem Real-Time-Messaging und belastet das Netzwerk zu stark. Abgesehen davon ist es sehr schwer, das richtige Intervall zu wählen. Wenn das Intervall zu kurz ist, ist die Belastung des Netzwerkes zu groß. Außerdem könnte dieses Verfahren für eine hohe Anzahl an Partizipanten vermutlich ungeeignet sein.

Um dieses Problem zu lösen, bietet SignalR  eine Lösung an, die immer funktionieren kann: Das sogenannte Long-Polling. Dabei werden weniger Requests zum Server geschickt, weil die Intervalle zwischen jedem Requests größer sind (z.B. 2 Minuten). In der Zwischenzeit bleibt eine Verbindung offen. Über diese kann der Server den Client erreichen. Dies scheint eine intelligente Lösung zu sein.

Allerdings könnte sie neue Probleme mit sich bringen: Die Web-Infrastruktur ist auf Request/Response ausgelegt. Das heißt, Load-Balancer, Proxies usw. routen in der Regel die Requests nach verschiedenen Verfahren zum Empfänger, beispielsweise um die Last besser zu verteilen. Im Falle von SignalR und ebenso WebSockets hält der Server eine Verbindung jedoch auch dann offen,  wenn keine Kommunikation erforderlich ist, und „torpediert“ so die Funktionsweise der konventionellen Web-Infrastruktur.

Bei einem Short Polling wäre dies nicht der Fall. Momentan kann man sogar davon ausgehen, dass für eine solche Art der Kommunikation dedizierte Server verwendet werden sollten. Hier gibt es jedoch bisher nur wenige Erfahrungen.

## Server Komponenten von SignalR

Die Low-Level-Komponente ist SignalR Core und implementiert in der Assembly SignalR.Server alle nötigen Server-Funktionalitäten. Darüber hinaus gibt es drei Host-Komponenten:

**SignalR.Hosting.AspNet**

Sie bietet die Klassen an, die für das Hosting eines SignalR Server in ASP.NET gebraucht werden.

**SignalR.Hosting.Owin**

Host-Implementierung des  Open-Webserver-Interface für .NET. Hierbei handelt es sich um den Versuch einen Standard für die Kommunikation zwischen .NET, Webserver und Web-Anwendung zu etablieren. Dieser Standard soll helfen, die Anwendung vom Server zu entkoppeln. Mehr dazu unter [3].

**SignalR.Hosting.Self**

Dieser Host wird verwendet, wenn der SignalR Server in einer Anwendung gehostet  werden soll, die kein Web-Server ist. Das könnte beispielsweise ein Office Add-In oder ein Windows Service sein.

## SignalR.Hosting.Self:

Dieser Host wird verwendet, wenn der SignalR Server in einer Anwendung gehostet  werden soll, die kein Web-Server ist. Das könnte beispielsweise ein Office Add-In oder ein Windows Service sein.

```
class Program
 {
    static void Main(string[] args)
    {
       string url = "http://localhost:8081";

       using (WebApplication.Start<Startup>(url))
       {
          Console.WriteLine("SignalR Host started on {0}", url);
          Console.ReadLine();
       }
    }
 }
```

In der Regel bietet SignalR zwei Modelle an: Peristed Connections und Hubs. Damit befassen wir uns später etwas ausführlicher.

## Client Komponenten von SignalR

Für die Implementierung der Client-Funktionalität bietet die SignalR- Bibliothek folgende Komponenten an:

**SignalRJs**

Die JavaScript-Bibliotheken

**SignalR.Client**

Die .NET Client- Bibliotheken, inklusive WinRT

**SignalR.Client.WP7**

Die SignalR Client-Bibliotheken für Windows Phone

**SignalR.Client.Silverlight und SignalR.Client.Silverlight5**

Die SignalR Client-Bibliotheken für Silverlight

## Persistent Connection Programmiermodell

**Client**

Soweit Ihre Anwendung nur einfache Kommunikation benötigt, wird Ihnen die Implementierung eigener Connection-Klassen wahrscheinlich genügen. Das Schwierige dabei ist zu wissen, was „einfach“ ist. In diesem Kontext scheint einfache Kommunikation vorzuliegen, wenn die Anwendung eine oder zwei (wenige) semantisch verschiedene, einfache Messages benötigt, um eine Funktionalität abzubilden.

Folgendes Beispiel zeigt clientseitig, wie die Connection im JavaScript erzeugt wird, wie die Messages empfangen werden und wie sie versendet werden.

```
$(function () {
    var connection = $.connection('/echo');


    connection.received(function (data) {
        $('#results).append('<li>' + data + '</li>');
```

```
    });

    connection.start().done(function() {
        $("#btnBroadcast").click(function () {
            connection.send($('#msg').val());//Bsp. 1.
            connection.send ({„prop1“: ‚value of prop1‘});//Bsp. 2.
        });
    });
});
```

Der Befehl *Send* ist keine Message, sondern zeigt, dass Eingabeparameter als Message zu verstehen sind. Dieser Befehl sorgt dafür, dass die Instanz eines anonymen Typs entsprechend serialisiert wird.

Um anonyme Typen mit den Eigenschaften Prop1 und Prop2 zu verarbeiten, benötigt man in SignalR nicht mehr als eine sehr einfache Implementierung der Basisklasse *PersistentConnection*. Dazu gleich etwas mehr.

Sollte Ihre Anwendung viele Typen solcher Messages verwenden, wird es schwierig, diese in einem Handler auseinander zu halten. In solchen Fällen sind die sogenannten Hubs besser geeignet. Dies ist die andere Art, einen Server zu implementieren, die im nächsten Kapitel detailliert beschrieben wird.

**Server**

Die wichtigsten Teile der Klasse *PersistentConnection* sind in folgendem Listing zu sehen.

```
using System.Threading.Tasks;
using Microsoft.AspNet.SignalR;

public class MyConnection : PersistentConnection
{
    protected override Task
    OnReceived(IRequest request,
            string connectionId,
            string data)
    {
        // Broadcast Daten zu alle clients
```

```
        return Connection.Broadcast(data);
    }

    protected override Task OnConnected(IRequest request, string connectionId)
    {
        return Connection.Broadcast("Connection " + connectionId + " connected");
    }

    protected override Task OnReconnected(IRequest request, string connectionId)
    {
        return Connection.Broadcast("Client " + connectionId + " re-connected");
    }

     protected override Task OnDisconnected(IRequest request, string connectionId)
    {
        return Connection.Broadcast("Disconnected " + connectionId );
    }

}
```

Im Allgemeinen bietet die Klasse virtuelle Methoden wie *OnReceived*, *OnConnected* und *OnReconnected* an. Allein damit ist es möglich festzustellen, ob ein neuer Participant sich mit dem Server verbunden hat (*OnConnected*), oder wenn eine Message empfangen wurde (*OnReceived*).

Wichtig ist, dass jede Art von Message eine Verbindung mit dem Server darstellt. Das heißt, wenn *OnReconnected* aufgerufen wird, bedeutet das nicht, dass ein neuer Client das System betreten hat. Diese Methode wird bei jedem Request aufgerufen. Sie wird beispielsweise auch dann aufgerufen, wenn Polling-Requests von Clients im Hintergrund geschickt werden, um die Verbindung zu erneuern.

Manchmal möchten Sie die Messages außerhalb der Connection-Klasse versenden. Um dies zu bewerkstelligen gibt es die Klasse **GlobalHost**.

```
var context = GlobalHost.ConnectionManager.GetConnectionContext<MyEndPoint>();
context.Connection.Broadcast(message);
```

Darüber hinaus ist es ebenso möglich bestimmte Aufrufer einer Gruppe zuzuordnen. Folgendes Listing zeigt wie das geht.

```
protected override IList<string>
 OnRejoiningGroups(IRequest request, IList<string> groups, string connectionId)
{
   return groups;
}

 protected override Task OnReceived(IRequest request,
                          string connectionId, string data)
{
   . . .
   // Daten an eine Gruppe senden
   return Groups.Send(groupName, message);
}

protected override Task OnDisconnect(string connectionId)
{
   return Groups.Remove(connectionId, "...");
}
```

Es ist zu beachten, dass die Send-Methode ein Argument vom Typ Objekt erwartet:

```
public static class ConnectionExtensions
{
   public static Task Broadcast(this IConnection connection,
      object value,
      params string[] excludeConnectionIds);

   public static Task Send(this IConnection connection,
                     string connectionId,
                     object value,
                     params string[] excludeConnectionIds);
}
```

Das gibt die Möglichkeit komplexere Messages (im XML-Jargon *complex-type*) ohne viel Mühe auszugeben:

```
Send({Prop1:‚xyz', Prop2:123});
```

Wenn Sie die Messages an die Gruppen verschicken möchten, können Sie ebenso **GlobalHost** verwenden.

```
var context = GlobalHost.ConnectionManager.GetConnectionContext<MyEndPoint>();
        context.Groups.Send(group, message);
```

JavaScript-Clients werden diese Objekte als JSON empfangen.
Das ist in der Tat sehr einfach und irgendwie cool. Allerdings gilt solche Vorgehensweise in der Welt von SOA als „Communication Anti-Pattern". Wenn Sie eine App entwickeln, die kommunikations-technisch eine oder zwei Messages bearbeitet, ist diese Technik in Ordnung. Wenn Sie aber viele Messages bearbeiten, sind Sie der eigentliche Service-Anbieter (private oder public). In diesem Falle sollten die Contracts klar definiert werden.

Bitte grundsätzlich keine anonymen Typen „auf Teufel komm raus" verwenden, nur weil sie schneller zu schreiben sind. Denken Sie daran, dass eine Anwendung nicht nur implementiert, sondern auch gewartet werden muss!

Last but not least: Ihre Implementierung der *PersistentConnection* ist eine gewöhnliche Klasse, die z.B. im Falle von ASP.NET Hosting irgendwo im Projekt existieren sollte. Darüber hinaus müssen Sie dem Host (z.B. IIS) die Route mitteilen, an der das Listening stattfinden wird. Dies geschieht üblicherweise in der *Application_Start*-Routine Ihrer Web-Anwendung, die SignalR anbietet:

```
public class Global : System.Web.HttpApplication
{
    void Application_Start(object sender, EventArgs e)
    {
        RouteTable.Routes.MapConnection<MyEndPoint>("echo", "/echo");
    }
}
```

Mehr über *PersistentConnection* erfahren Sie unter [4].
Wenn Sie mehr über die JavaScript-Bibliothek und PersistentConnections erfahren möchten, folgen Sie dem Verweis[5].

## Hubs

Wenn Ihre Anwendung hauptsächlich serviceorientiert arbeiten soll, sind Hubs die bessere Wahl gegenüber *PersistentConnections*. Ein Hub ist eine Klasse, die von der Basisklasse *Hub* ableitet. Das Interessante an diesem Konzept ist, dass die Basis-Klasse Hub keine virtuellen Methoden  oder Schnittstellen vorsieht, die man überschreiben bzw. implementieren soll. Das wird deshalb betont, weil diese Vorgehensweise in fast allen APIs, die vergleichbare Aufgaben lösen, üblich ist.
Eine vollständige Implementierung eines Hub ist in diesem Listing zu sehen:

```
public class MyHub : Hub
{
    public void DoWork(MyEntity input)
    {
        Clients.All.onMessage (new {. . .});
    }
}
```

Die Methode d*oWork* in der Klasse *MyHub* ist quasi die Funktion die remote aufgerufen wird. In der WCF-Welt wäre dies eine gewöhnliche SOAP Operation.

Die Idee hinter diesem Konzept ist einem Client zu ermöglichen eine beliebige public-Methode an einem Hub aufzurufen. Das Beispiel oben zeigt wie die JavaScript Funktion *onMessage* auf allen Clients aufzurufen ist.
Folgender JavaScript Code zeigt wie dies in JavaScript zu realisieren ist:

```
var myHubProxy = $.connection.myHub;
    contosoChatHubProxy.client.onMessage = function (msg) {
    console.log(msg);
};

 $.connection.hub.start().done(function () {
```

```
    $('#newContosoChatMessage').click(function () {
        contosoChatHubProxy.server.doWork(msg)
            .done(function(result){
            console.log(result);
        })
        .fail(function(e1,e2,e3)
        {
        })
    });
});
```

Wenn JavaScript die Funktion *doWork* aufruft, wird im Service die Methode MyHub.DoWork (MyEntity input) aufgerufen. Die Message wird, soweit kompatibel definiert, automatisch in die Instanz von *MyEntity* umgewandelt.

Zugegebenermaßen ist dies für Entwickler sehr einfach zu implementieren, dank einer Proxy-Klasse. Beim Laden der Web-Anwendung sollte folgender Skriptverweis eingebunden werden:

```
<script src="/Daenet.SignalR/signalr/hubs" type="text/javascript">
</script>
```

Durch diesen Request erzeugt SignalR im Hintergrund dynamisch eine JavaScript-Datei, die einige Hilfsfunktionen bietet.

Nachdem JavaScript den Aufruf zu *doWork* abgesetzt hat, blockt dieser Call, solange die Ausführung in MyHub.DoWork andauert, nicht. Das heißt, die Funktion *doWork* ist clientseitig asynchron, unabhängig davon, ob sie Rückgabeparameter hat oder nicht.

## Unter der Haube

SignalR unterstützt verschiedene Protokolle (Transports), die den meisten Entwicklern kaum bekannt sind: WebSockets, LongPolling, ForeverFrames und ServerSentEvents.
Wenn clientseitig beispielsweise

```
connection.start();
```

aufgerufen wird, versucht SignalR das beste Protokoll zu finden. Im Idealfall wäre dies der WebSocket-Transport. Sollte dies nicht möglich sein, wird SignalR für den Falle, dass der Internet Explorer verwendet wird, versuchen die Verbindung mit dem Server über ForeverFrame-Transport aufzubauen. Im Falle von Firefox, Chrome, Opera und Safari wird SignalR sein Glück mit dem ServerSentEvents-Transport versuchen. Wenn keine der Varianten in Frage kommt, gibt es eine Fallback-Lösung namens LongPolling. Bitte beachten Sie, dass LongPolling die einzige Variante ist, die eine Annäherung an Real-Time-Messaging ermöglicht. Selbst im Falle von LongPolling sind extrem gute Echtzeitwerte möglich, weil die Kommunikation eigentlich durch die geöffneten Streams stattfindet.

Unabhängig vom Transport, wird eine Message vom Client immer einen Request/Response-Aufruf initiieren. Darüber hinaus besteht zusätzlich eine offene Verbindung zwischen dem Client und dem SignalR-Service (Persistent Connection bzw. Hub). Je nach Transport ist die Verbindung unterschiedlich aufgebaut. Auch das Format der Daten, die zwischen Client und Service hin und her fließen, ist logischerweise protokollabhängig.

Abb. 2 zeigt ein Long-Polling. Dies passiert in der Regel, wenn die Infrastruktur Proxies enthält, die kein Streaming unterstützen. Zum Beispiel unterstützt der Debugging-Proxy Fiddler auch kein Streaming in der Default-Konfiguration.

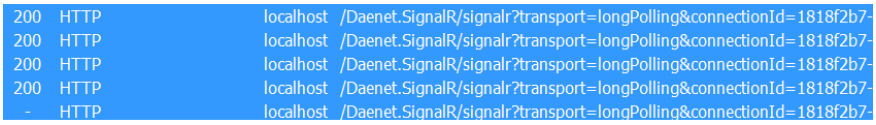



*Abb. 2 Long-Polling*

Eine andere Möglichkeit einen Transport zu erzwingen, ist beim Aufrufen der Startmethode die Property *transport* zu setzen. Folgender Aufruf erzwingt den LongPolling-Transport:

```
connection.start({ transport: 'longPolling' });
```

Man kann auch mehrere Transports kombinieren:

```
connection.start({ transport: 'foreverFrame longPolling');
```

Wenn LongPolling aktiv ist, hält die Verbindung ca. 2 Minuten, insoweit keine Messages von Service zum Client geschickt wurden. Wenn eine Message zum Client geschickt wurde, erneuert der Client die Verbindung.
Wenn ein Client eine Message zum Service sendet, wird immer ein neuer Request zum Service geschickt.

Ähnlich wie beim LongPolling zeigen die Abbildungen 3 und 4 wie dieses Szenario mit den Transportarten ForeverFrames und ServerSentEvent aussieht.

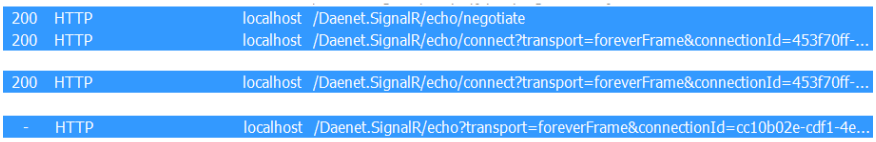



*Abb. 3 Forever Frames*

Im Falle von ForeverFrames und ServerSentEvents wird eine Verbindung aufgebaut, die alle 2 Minuten erneuert wird, wenn keine Messages zwischen Client und Service fließen. Messages vom Service zum Client fließen durch eine geöffnete Verbindung und verursachen keinen neuen HTTP-Requests. Wenn ein Client eine Message zum Service schickt, wird ein neuer Request erzeugt.

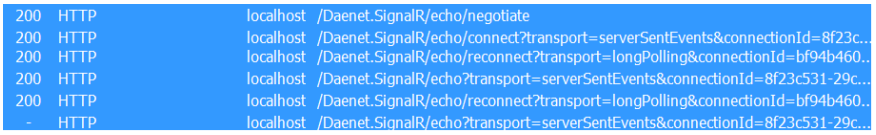



*Abb. 4 ServerSentEvents*

## Fazit

In diesem Artikel haben wir gesehen, wie man asynchrones Real-Time- Messaging über HTTP mit SignalR aufbauen kann. Gegenüber WebSockets hat SignalR den Vorteil, dass man nicht auf den Support von WebSockets in HTML5 warten muss.

Auf der anderen Seite ist die Entwicklung von WebSockets tief in der Microsoft-Plattform verzahnt und hat schon heute einen festen Platz in der Windows Communication Foundation Strategie. Dies ist der Vorteil von WebSockets gegenüber SignalR.

SignalR wird vermutlich mittelfristig seinen berechtigten Platz im ASP.NET Open-Source finden.

Die neuen Push-Konzepte, egal ob SignalR oder WebSockets, sind sehr innovativ und bieten zweifellos eine Plattform für bisher schwer denkbare Szenarien an. Allerdings endet diese Reise hier bestimmt nicht. Heute wissen wir noch nicht, wie die herkömmliche Infrastruktur (Netzwerke, Router, LoadBalancer und Server) auf diese neue Art von Anwendungen wirklich reagieren wird.

Bitte vergessen Sie nicht, dass alle aufgezählten Komponenten für Request/Response konzipiert sind. Auch in diesem Anwendungsbereich gibt es noch ungelöste Probleme. Wie verschickt man beispielsweise einen Broadcast, wenn die Services auf mehreren Nodes im Cluster laufen?

Diesbezüglich gibt es schon Versuche mit dem Einsatz von ServiceBus Topics. Das ist jedoch ein Thema, dass den Rahmen dieses Artikels sprengen würde.

[1] WebSocket Spezifikation

http://websocket.org

[2] SignalR auf GitHub

https://github.com/SignalR/SignalR/wiki/Getting-Started

[3] OWIN Spezifikation

http://owin.org/spec.html

[4] PersistentConnections

https://github.com/SignalR/SignalR/wiki/PersistentConnection

[5] JavaScript-Client und PersistentConnections

https://github.com/SignalR/SignalR/wiki/SignalR-JS-Client

# Kapitel 5: Mobile Development

## Patric Schouler (SDX AG) – Near Field Communication mit Windows Phone

### Arbeitgeber

• SDX AG

• Chief eXpert (.NET)

### Technologieschwerpunkte

• Web- sowie Windows Phone-Entwicklung

• SDX Service-Leistungen Agile Entwicklung (Scrum), Architekturberatung, Coaching und Lead Development

### Mein Urlaubstipp

Eines meiner liebsten Urlaubsziele ist Mittelschweden und hier insbesondere die Stadt Stockholm. Faszinierend an dieser Stadt ist die schöne Mischung von Inseln, Wasser, Natur gepaart mit urbanem Lebensstil ohne die Hektik von anderen Metropolen zu verströmen. Ein ganz besonderes Highlight sind natürlich die Schären, welche eine ganz besondere Magie besitzen. Meine Empfehlung: Einmal ein paar Nächte auf einer kleinen Schären-Insel verbringen. Möchte man dann fast unberührte Natur erleben, findet man dies unweit von Stockholm rund um den Mälarsee.

### Meine Inseltechnologie

Im letzten Jahr hat mich am meisten das überarbeitete Windows Phone 8 beeindruckt. Diese Neuauflage hat nicht nur viele Verbesserungen für die Benutzung gebracht, sondern durch die Kooperation mit Nokia ist es auch gelungen, konkurrenzfähige Hardware zu iPhone und Co zu etablieren. Als Entwickler hat mich positiv überrascht, dass man eine starke Kompatibilität zu Windows Phone 7 erreicht hat und das viele Kritikpunkte aus der Developer Community in Windows Phone 8

umgesetzt wurden. Die innovativste Technologie in diesem Zusammenhang ist für mich die Unterstützung der Near Field Communication (NFC) bei Windows Phone.

## Near Field Communication mit Windows Phone

Near Field Communication (NFC) ist ein internationaler Standard zum kontaktlosen Austausch von Daten per Funktechnologie über kurze Strecken von bis zu 10 cm mit einer Datenübertragungsrate von maximal 424 kBit/s. Obwohl die Entwicklung der Technologie bereits ihre Anfänge in 2002 hatte, steckt ihre Verbreitung bisher noch in den Startlöchern. Die immer größer werdende Anzahl von NFC-fähigen Geräten - insbesondere Smartphones -, die vielfältigen Einsatzmöglichkeiten und die einfache Handhabung machen diese Technologie jedoch zunehmend interessanter für kommerzielle Anwendungen.

### Etwas Historie

Die NFC Technologie basiert auf die bereits im zweiten Weltkrieg eingesetzte RFID-Technologie (radio-frequency identification). Hier wurden Transponder und Lesegeräte an Panzern und Flugzeugen eingesetzt, um fremde von eigenen Fahrzeugen unterscheiden zu können. In den 70igern gab es dann die ersten Systeme zur Sicherung von Waren auf RFID-Chip Basis. Ab 1990 wurden große Fortschritte in der technologischen Entwicklung von Transpondern erzielt, sodass immer mehr Anwendungen entwickelt wurden. Darunter zählen Zugangskontrollen zu Gebäuden, die Wegfahrsperre in Autos, Mautsysteme oder die Zeiterfassungen in Betrieben und die Zeitmessung bei Sportveranstaltungen.

NFC wurde im Jahre 2002 durch die beiden Firmen NXP Semiconductors und Sony entwickelt, die führend im Bereich von kontaktlosen Chipkarten sind. Zwei Jahre später wurde durch diese beiden Firmen und Nokia das NFC Forum gegründet. Dieses hat es sich zur Aufgabe gemacht, die Entwicklung von NFC voranzutreiben und weiter zu standardisieren. Die ersten NFC-fähigen Handys gab es bereits 2005, mit denen ein Feldversuch in Frankreich durchgeführt wurde. Hier konnten Benutzer Waren im Einzelhandel bezahlen, Parktickets kaufen oder touristische Informationen abrufen. In Deutschland wurde ebenso 2005 ein Feldversuch für den Personennahverkehr vom Rhein-Main-Verkehrsverbund (RMV) gestartet, bei dem Fahrscheine über ein Handy gekauft werden konnten.

### Wo wird NFC genutzt?

Bisher kommt diese Technik vor allem in Lösungen für das Micropayment - bargeldlose Zahlungen kleiner Beträge – zum Einsatz. In Deutschland wird die Technik beispielsweise von den Sparkassen, unter dem Namen *girogo* zur Zahlung von Summen bis zu 20 Euro angeboten. Des Weiteren bietet die

Deutsche Bahn mit dem Touch & Travel-System ein sehr einfaches Verfahren zum Erwerben von Fahrtickets mit NFC-fähigen Smartphones, welches für alle Fernverkehrs-, einigen Auslandsverbindungen und in ausgewählten Städten und Regionen verfügbar ist.

Ein weiteres Anwendung Szenario sind intelligente Plakate, denen zusätzliche technische Informationen hinzugefügt wurden. Dies geschieht heute noch in der Regel mit QR-Codes, kann aber mittels NFC-Technologie um weitere Möglichkeiten, wie die Übertragung von Texten in mehreren Sprachen, Bilder und Aktionen erweitert werden. Darüber hinaus bieten NFC Tags hier einen besseren Schutz gegen „Edding Attacken" die dann QR Codes unlesbar machen. Noch im Versuchsstadium befindet sich aktuell die Idee einiger Automobilhersteller (Hyundai, BMW), mit Hilfe des Smartphones und NFC-Funktionen die Autotür zu entriegeln und persönliche Einstellungen der Komfortoptionen im Automobil (Sitz- und Spiegelpositionen, Senderwahl) zu aktivieren.

Dies sind an dieser Stelle nur ein paar Anwendungsfälle. In Zukunft werden diese, durch die weitere Verbreitung der NFC Technologie, sicherlich noch zunehmen.

Aktuell unterstützen u.a. Geräte von Nokia, Samsung, Blackberry, Google, HTC, LG, Motorola und Sony die NFC Verfahren. Nachlegen wird hier sicher auch bald Apple, da das aktuelle iPhone 5 die NFC-Technologie noch nicht beinhaltet. Neben vielen aktuellen Smartphones sind auch einige Tablet-PCs mit eingebauter NFC-Technik verfügbar.

## NFC Kommunikationswege

NFC setzt auf induktiv gekoppelte Systeme, bei denen passive Transponder die Energie aus einem Magnetfeld ziehen. Da jedoch bei NFC, im Gegensatz zu RFID, die strikte Trennung zwischen Lesegerät und Transponder entfällt, ergeben sich einige neue zusätzliche Kommunikationsmöglichkeiten. Für diese unterschiedlichen Kombinationen von Komponenten sind verschiedene Übertragungsmodi vom NFC Forum definiert worden.

So ermöglicht der *Reader/Writer-Modus* die Kommunikation zwischen einem NFC-Gerät und einem passiven Transponder. Im *Peer-to-Peer*-Modus können zwei NFC-Geräte Daten untereinander austauschen (z.B. bei Windows Phone Tap & Senden). Der *Card-Emulation-Modus* ermöglicht dem NFC-Gerät als Smartcard zu agieren und als passiver Teilnehmer mit einem normalen RFID basierenden Lesegerät zu kommunizieren.

## Was braucht man alles um mit der NFC Entwicklung loszulegen?

Zur Entwicklung von NFC Anwendungen für Windows Phone benötigen wir Visual Studio 2012, das Windows Phone SDK und ein Windows Phone 8 (z.B. Nokia Lumia 920). Man braucht ein echtes Gerät, da vom aktuellen Windows Phone Emulator NFC leider noch nicht unterstützt wird. Als weitere Software-Komponente benötigt man die NDEF Library for Proximity APIS, da die native Windows Phone API keine NDEF Nachrichten schreiben kann. Es ist auch möglich, die NDEF Library via Nuget mit dem Kommando *Install-Package NdefLibrary* zu installieren.

Als weiteres benötigen wir natürlich noch brauchbare NFC Tags, welches sich bei mir als ein sehr schwieriges Unterfangen herausgestellt hat. Es waren zwei Bestellversuche notwendig, um geeignete NFC Tags zu erhalten. Wichtig ist nämlich hierbei, dass man keine leeren NFC Tags verwendet, sondern NFC Tags, welche mit dem NDEF (*NFC Data Exchange Format*) Format beschrieben bzw. formatiert wurden. Aktuell bietet uns die Windows Phone Proximity Api keine Möglichkeit NFC Tags zu formatieren. Ein NFC Tag kann in verschiedenen Ausprägungen – als Armband, Schlüsselanhänger oder Papier Sticker – bezogen werden. Einfach NFC Tags in eine Suchmaschine seiner Wahl eingeben und einen geeigneten Shop suchen.

Ein NFC Tag kann durch das Windows Phone mit unterschiedlichen Informationen im NDEF Format beschrieben werden. Die Größe der aktuellen Tags reicht von 168 Byte, über einem bis zu maximal vier Kilo Byte.

Unterstützt werden von Windows Phone folgende NFC Tags:

- Type 1: Topaz family
- Type 2: Mifare Ultralight family, my-d-move, NTag
- Type 3: Felica family
- Type 4: Desfire family
- Non standardized: Mifare Standard

## Einige Einschränkungen

Aktuell müssen wir beim Umgang mit NFC Tags mit einigen Einschränkungen leben, die die Arbeit mit NFC Tags eigentlich nicht beschneiden, sondern lediglich einige Anwendungsszenarien beschränken.

- Es ist durch die fehlende Unterstützung des RAW Modus bei Windows Phone nicht möglich, den Schreibschutz eines NFC Tags zu aktivieren.
- Mittels Windows Phone können keine NFC Tags formatiert werden – also nur vorformatierte oder beschriebene Tags im NDEF Format kaufen. Eine weitere Alternative ist die Verwendung eines NFC USB Reader/Writer oder – ich wage es kaum zu erwähnen – die Benutzung eines Android Geräts.
- Der NFC Tag darf nur NDEF Nachrichten enthalten, damit er lesbar bleibt.
- Es ist nicht möglich, den kompletten Speicherplatz eines NFC Tags zu benutzen. Beispielsweise können nur 716 Bytes bei einem 1KB Tag oder 454 Bytes bei einem Tag mit einer Größe von 512 Byte genutzt werden
- Die Proximity API von Windows Phone bietet keine Möglichkeit direkt NDEF Nachrichten auf den Tag zu schreiben. Dies ist aber durch die Benutzung der Open Source NDEF Library möglich, die ich in meinem Beispiel auch verwendet habe
- Es ist nicht möglich die Daten eines NFC Tags zu empfangen oder zu schreiben, wenn die Windows Phone App im Hintergrund läuft. Man kann aber über Schemata-Zuordnungen und die Auto-Launch Funktion – dazu später mehr - eine im Hintergrund oder noch nicht laufende Applikation starten

## Der Schüssel zu allem: Windows.Networking.Proximity

Im Namensbereich Windows.Networking.Proximity befindet sich der Einstiegspunkt der Proximity API von Windows Phone. Die wichtigste Klasse ist hierbei *ProximityDevice*.

## Der Use Case - Check-in mit NFC Tags

Innerhalb der SDX AG nutzen wir eine neue eigenentwickelte Zeiterfassungs-Software mit der Bezeichnung WorkTime. Dieses System basiert auf dem Microsoft CRM System und ist als Client für Windows 8 (Modern UI App), Windows Phone 8 sowie als Web-Anwendung verfügbar. Also warum sich nicht die NFC Tags zu Nutze machen und die lästige Zeiterfassung bei unterschiedlichen Projekten dadurch verkürzen, das ein NFC Tag die projektspezifischen Daten zu einem Zeiterfassungsvorgang bereits enthält und durch das „Tappen“ mit dem Windows Phone ein Zeiterfassungsvorgang automatisiert vorgenommen werden kann. Ich hätte dann für unterschiedliche Kunden und Projekte jeweils einen anderen farblich gekennzeichneten Tag und könnte diese sogar am entsprechenden Arbeitsplatz irgendwo hin kleben.

Also Idee in die Tat umsetzen…

## Erster Schritt – Speichern der Daten auf dem NFC Tag

Das folgende Code Schnipsel verwende ich, um den serialisierten aktuellen Zeiteintrag auf einem NFC Tag zu speichern. Wichtige Einstiegspunkte sind hierbei die Events *DeviceArrived* und *DeviceDeparted*, welche entsprechend geworfen werden, sobald sich ein NFC Tag in der Nähe des Windows Phone befindet oder sich wieder entfernt hat. Im Konstruktor meiner NfcService Klasse ermittele ich mir dazu zunächst das aktuelle Proximity Device über die *ProximityDevice.GetDefault()* Methode.

Zum Zugriff auf den NFC Tag verwende ich in meinem Beispiel, wie gesagt, die NDEF Library welche es auf einfache Art und Weise ermöglicht NDEF Datensätze zu parsen sowie diese zu erstellen. Ein NFC Tag kann dabei immer einen oder mehrere NDEF Datensätze enthalten. Die NDEF Library unterstützt verschiedene Datensatz Typen, wie beispielsweise *WpSettings* (zum Starten von Windows Phone Einstellungsseiten), *Mailto* (Datensätze zum automatischen Versenden von E-Mails), *DriveTo* und *WalkTo* (Starten der Navigationsfunktionen von Windows Phone 8), *Nokia Accessories* (Datensatz zum Starten von Nokia Zusatzanwendungen auf Lumia Smartphones) und der wichtige *Launch App Record* zum automatischen Start einer Windows Phone App. Dieses NDEF Nachrichtenformat wird auch in folgendem Beispiel verwendet.

```
private const string ProductId = "{Your ProductId from WMAppManifest.xml}";
private readonly ProximityDevice _device;
private string _messageContent;

public NfcService()
{
    _device = ProximityDevice.GetDefault();
}

#region public events

public event EventHandler NfcNotAvailable;

protected virtual void OnNfcNotAvailable()
{
    EventHandler handler = NfcNotAvailable;
    if (handler != null) handler(this,EventArgs.Empty);
}

public event EventHandler<string> NfcArrived;

protected virtual void OnNfcArrived(string e)
{
    EventHandler<string> handler = NfcArrived;
```

```
    if (handler != null) handler(this, e);
}

public event EventHandler<string> NfcDeparted;

protected virtual void OnNfcDeparted(string e)
{
    EventHandler<string> handler = NfcDeparted;
    if (handler != null) handler(this, e);
}

public event EventHandler NfcMessageWritten;

protected virtual void OnNfcMessageWritten()
{
    EventHandler handler = NfcMessageWritten;
    if (handler != null) handler(this, EventArgs.Empty);
}

#endregion

public static string NfcLaunchParameterName
{
    get { return "ms_nfp_launchargs"; }
}

public void WriteLaunchRecordToNfc(string messageContent)
{
    // check if NFC is available on the Windows Phone device
    if (_device != null)
    {
        _device.DeviceArrived += DeviceArrived;
        _device.DeviceDeparted += DeviceDeparted;
        _messageContent = messageContent;
    }
    else
    {
        OnNfcNotAvailable();
    }
}

void DeviceDeparted(ProximityDevice sender)
{
    var deviceId = String.Empty;
    if (sender != null)
    {
        deviceId = sender.DeviceId;
    }

    OnNfcDeparted(deviceId);
}
```

```
void DeviceArrived(ProximityDevice sender)
{
    var deviceId = String.Empty;
    if (sender != null)
    {
        deviceId = sender.DeviceId;
    }

    OnNfcArrived(deviceId);

    // Create a LaunchApp record, specifying our recognized text as arguments
    var record = new NdefLaunchAppRecord { Arguments = _messageContent };
    // Add the app ID of your app!
    record.AddPlatformAppId("WindowsPhone", ProductId);

    // Wrap the record into a message, which can be written to a tag
    var msg = new NdefMessage { record };

    _device.PublishBinaryMessage("NDEF:WriteTag", msg.ToByteArray().AsBuffer(),
MessageWrittenHandler);

}

private void MessageWrittenHandler(ProximityDevice sender, long messageId)
{
    // detach event handlers
    _device.DeviceArrived -= DeviceArrived;
    _device.DeviceDeparted -= DeviceDeparted;

    // stopping publishing the message
    _device.StopPublishingMessage(messageId);

    OnNfcMessageWritten();
}
    }
```

Beim Schreiben auf einen NFC Tag sind folgende Punkt besonders zu beachten:

- Bei Verwendung eines NdefAppLauchRecord muss als Plattform *WindowsPhone* und die ProductId der App aus dem AppManifest angegeben werden
- Ein Aufruf von *_device.StopPublishingMessage* nach erfolgtem Schreiben ist notwendig
- Die Message Größe des NDEF Records darf nicht die Nettokapazität des NFC Tags überschreiten
- Um NFC in der App nutzen zu können, muss *ID_CAP_PROXIMITY* in den Application Capabilities aktiviert werden

Der Umgang mit den anderen NDEF Nachrichtentypen ist analog zu diesem Beispiel, außer dass hier die Nachricht anders aufgebaut werden muss.

So weit so gut - jetzt können wir die Daten auf einen NFC Tag schreiben. Aber wie schaffe ich es, dass meine App nun automatisch beim „Tappen“ mit meinem Lumia gestartet wird?

## Hinzufügen der Auto-Start Funktionalität

Um eine Windows Phone 8 automatisch starten zu können, ist es notwendig eine URI Zuordnung für die jeweilige App zu erstellen. Was diese URI auszeichnet, ist das diese mit einem festen Schemanamen beginnt, der für diese App registriert wurde. Nachdem diese Registrierung dem Betriebssystem bekannt gemacht wurde, wird durch das „Tappen“ mit einem NFC Tag, welcher einen LaunchApp NDEF Record enthält, über die ProductId im NDEF Record genau eine solche URI von Windows Phone OS aufgebaut und die App mit dieser URI gestartet.

Die Nutzung von Schema-Zuordnungen zu einer Windows Phone App ist übrigens ein allgemeines Feature und kann beispielsweise auch zum Aufruf einer Windows Phone App über eine URI aus einer anderen App genutzt werden.

Beispiel: *worktime:ShowTimeSheetEntry?UniqueId=89819279-4fe0-4531-9f57-d633f0949a19*

In diesem Beispiel ist *worktime* das registrierte App Schema und alles nach dem Doppelpunkt kann durch die App in der UriMapper Klasse – dazu später mehr – interpretiert werden.

Über den folgenden Code kann die WorkTime App aus einer anderen App aufgerufen werden:
Windows.System.Launcher.LaunchUriAsync(new System.Uri("*worktime:ShowTimeSheetEntry?UniqueId=89819279-4fe0-4531-9f57-d633f0949a19*"));

Um eine URI Verknüpfung für eine App zu erstellen, muss die *WMAppManifest.xml* Datei des Windows Phone Projekts mit dem XML(Text) Editor bearbeitet werden. Im *Extensions*-Element der Manifest-Datei wird eine URI-Verknüpfung durch das Hinzufügen eines *Protocol*-Elements definiert. Bitte beachten, dass das Extensions Element **sofort nach dem Tokens-**Element eingefügt werden muss. Man kann maximal 10 URI-Verknüpfungen in jeder App definieren. Die Schema-Namen einer App müssen dabei eindeutig bei den installierten Apps sein – also genau überlegen was man hier nimmt.

```
<Extensions>
  <Protocol Name="worktime" NavUriFragment="encodedLaunchUri=%s" TaskID="_default" />
</Extensions>
```

Einige Schemata sind reserviert und würden bei Verwendung ignoriert werden. Für mehr Informationen, siehe Reservierte Datei und URI Verknüpfungen für Windows Phone 8.

Die generierte URI aus dem NFC Tag LaunchApp Record zum automatischen Starten der WorkTime App sieht dann folgendermaßen aus:

Protocol?encodedLaunchUri=worktime:<xml string of our TimesheetEntry object>

Hinter dem Doppelpunkt folgt die XML Repräsentanz der Zeiteintrags-Instanz.

## Extrahieren der Aufrufparameter aus der URI

Um nun die aufrufende URI aufzulösen, ist die Klasse *UriMapper* zu implementieren, welche von *UriMapperBase* abgeleitet ist. Die Argumente der LaunchApp NDEF Nachricht auf unserem Tag müssen dann in dem Parameter *ms_nfp_launchargs* für den Aufruf der gewünschte Einstiegsseite der Windows Phone App übergeben werden, da dieser Mechanismus auch vom Betriebssystem beispielsweise über eine Sekundärkachel (*deep linking*) verwendet wird.
(Niemand weiß, ob *ms_nfc_launchargs* korrekt ist oder *ms_nfp_launchargs* weil in der MSDN Dokumentation werden beide erwähnt :-( - aber alle Beispiele funktionieren mit *ms_nfp_launchargs*)

Die folgende UriMapper Klasse dekodiert zunächst die aufrufende Uri und benutzt dann einen regulären Ausdruck um die Payload der NDEF Nachricht zu extrahieren:

```
class WorkTimeUriMapper: UriMapperBase
{
    public override Uri MapUri(Uri uri)
    {
        // Example: "Protocol?encodedLaunchUf64ri=worktime:testmessage"
        var tempUri = HttpUtility.UrlDecode(uri.ToString());
        var launchContents = Regex.Match(tempUri, @"worktime:(.*)$").Groups[1].Value;
        if (!String.IsNullOrEmpty(launchContents))
        {
            // Call MainPage.xaml with parameters
            return new Uri("/MainPage.xaml?ms_nfp_launchargs=" + launchContents, UriKind.Relative);
        }

        // Include the original URI with the mapping to the main page
        return uri;
    }
}
```

Um die neu definierte Klasse WorkTimeUriMapper nun als UriMapper zu verwenden, ist es notwendig, folgenden Anweisungen in die **InitializePhoneApplication** Methode der App.xaml.cs einzufügen.

```
RootFrame = new TransitionFrame();
RootFrame.UriMapper = new WorkTimeUriMapper();
RootFrame.Navigated += CompleteInitializePhoneApplication;
```

Jetzt wird die aufrufende Uri mit dem Schema der App in eine relative Adresse der gewünschten Aufrufseite (in meinem Fall MainPage.xaml) transformiert und die Payload der NDEF Nachricht wird im Parameter *ms_nfp_launchargs* übergeben.

## Parsen der Argumente und Zeiteintrag anlegen

In meiner MainPage.xaml, oder der jeweiligen Zielseite, müssen nur noch die Übergabeargumente extrahiert und einer entsprechenden Funktion übergeben werden. Im *OnNavigateTo* Event kann hierzu mittels *NavigationContext.QueryString["ms_nfp_launchargs"]* der Aufrufparameter, also die Payload des NFC Tags, ausgelesen werden. In meinem Fall handelt es sich, wie bereits erwähnt, um die serialisierte Klasseninstanz eines Zeiteintrags. Durch eine Funktion wird nun auf Basis dieses Zeiteintrags ein neuer Eintrag für den aktuellen Tag angelegt, den ich aus Gründen der besseren Usability noch mal zur Betätigung anzeige.

## Das Ergebnis

Schreiben eines NFC Tags über die Kontextfunktion aus einem bestehenden Zeiteintrag:


## Ziel erreicht

Jetzt kann ich diese wunderbar farbigen NFC Tags für die automatisierte Zeiterfassung in meinen unterschiedlichen Projekten nutzen:

## Weiterführende Links zum Thema

- Entry page for using NFC tags with Windows Phone 8 on Nokia developer Wiki
- Microsoft documentation about the Proximity API
- Common page of the Nokia developer Wiki about NFC technology
- Comparison about supported NFC technologies on the different platforms
- NDEF Open source library to read and write NDEF formatted NFC messages on Windows Phone 8

## Daniel Meixner (Microsoft) – Windows Phone 8 Cross Platform Development

### Arbeitgeber

• Microsoft Deutschland GmbH

### Technologieschwerpunkte

• Windows Phone App Development

• Windows 8 App Development

• Visual Studio

• Kinect for PC

### Mein Urlaubstipp

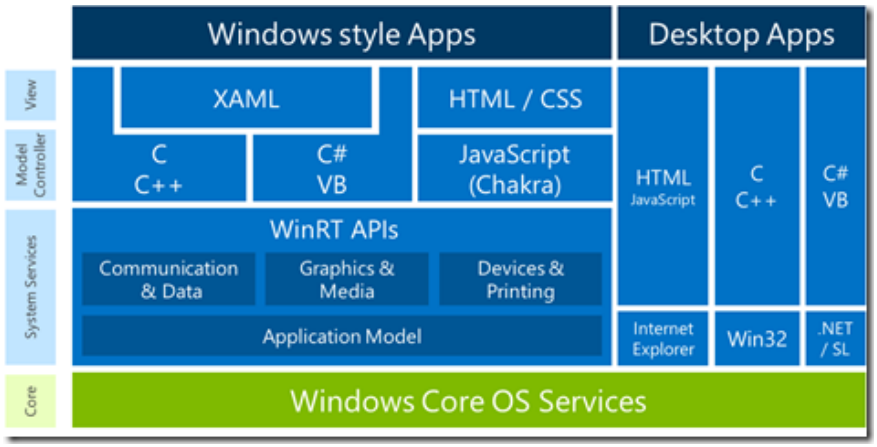
Mein Urlaubstipp ist Torri del benaco. Das ist ein kleines verschlafenes Nest direkt am Gardasee. Nicht ganz so überlaufen wie die bekannteren Touri-Orte in der Ecke und deshalb meine Empfehlung. Wenn man vor Ort ist, lohnt sich ein Fußmarsch nach Albisano, weil man von dort einen recht netten Ausblick auf den See hat. Außerdem gibt's dort Pizza und Rotwein. Was will man mehr?

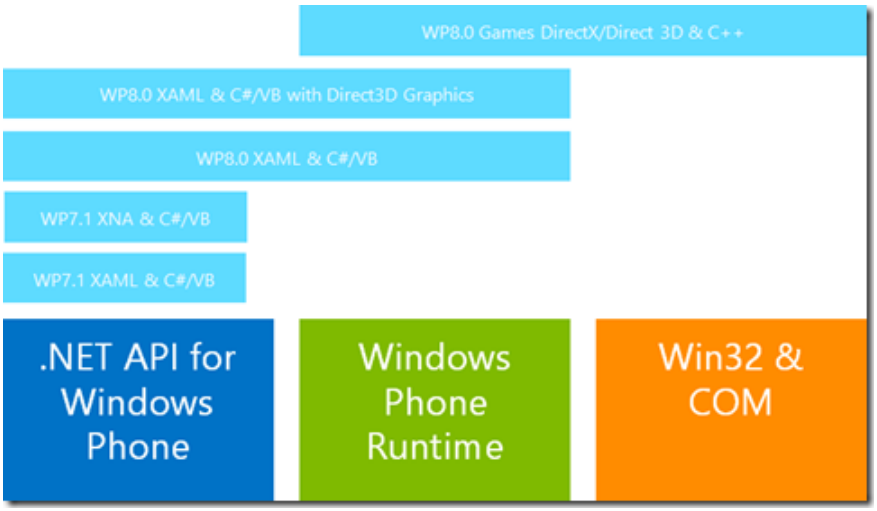
### Meine Inseltechnologie

Im letzten Jahr hat sich alles um Windows 8 und Windows Phone 8 gedreht. Ist doch klar, dass man sich da als .NET Entwickler Gedanken macht, wie man für beide Plattformen entwickeln kann, ohne alles doppelt schreiben zu müssen, oder?

## Windows 8 + Windows Phone 8 Cross Platform App Development mit C#/XAML (Teil 1)

Grundsätzlich ist das Ansinnen möglichst viel Code  wiederverwenden zu wollen absolut nachvollziehbar. Wenn wir einen Blick auf die – inzwischen wohl hinlänglich bekannten – Architekturdiagramme der beiden Betriebssysteme werfen, werden wir feststellen, dass es durchaus Bereiche gibt, die gleiche oder ähnliche Namen tragen und demzufolge doch Hoffnungen wecken, dass wir uns hier – für den Fall, dass wir eine App für WP8 als auch W8 entwickeln wollen – Arbeit sparen können. Und hier ist gleich ein ganz wichtiger Punkt festzuhalten: Wir müssen – da führt kein Weg vorbei – zwei separate Apps entwickeln. W8 und WP8 basieren zwar auf dem gleichen Betriebssystemkernel (was einige nette Nebeneffekte hat). Dennoch sind beides erst mal getrennte Betriebssysteme, die sich während der Entwicklung von Apps aber erstaunlich ähnlich anfühlen können. Unsere Aufgabe ist es also herauszufinden, wo diese Ähnlichkeiten liegen, um diese dann geschickt auszunutzen.



Für diesen Blogpost konzentriere ich mich auf den C#/XAML/Windows (Phone) Runtime Bereich in den Diagrammen. Für Cross-Plattform Entwicklung eignet sich grundsätzlich auch C++, in diesem Post spare ich dieses Thema allerdings aus. Man mag sich fragen, weshalb JS/HTML im Phone Diagramm gar nicht gelistet ist, wo es doch sogar Templates für WP8 App Entwicklung in Visual Studio für JS/HTML gibt. Der Hintergrund dafür ist, dass wir mit JS/HTML auf WP8 keine "nativen" Apps schreiben können, sondern diese immer in einem Webcontrol gehostet sein müssen. Bei W8 ist das anders, hier brauchen wir keinen Container, in dem das JS läuft, wir können direkt aus JS auf die (nativen) WinRT APIs zugreifen.

Verwirrend? Vielleicht! Einmal verstanden, weiß man aber, was geht und was nicht. Natürlich wäre es aus Entwicklersicht wünschenswert, wenn hier eine Vereinheitlichung der Programmiermodelle möglich wäre. Ich denke, dass wir mit C#/XAML wie ich es hier erläutern werde schon einen ordentlichen Weg einschlagen.

## Generelle Strategie

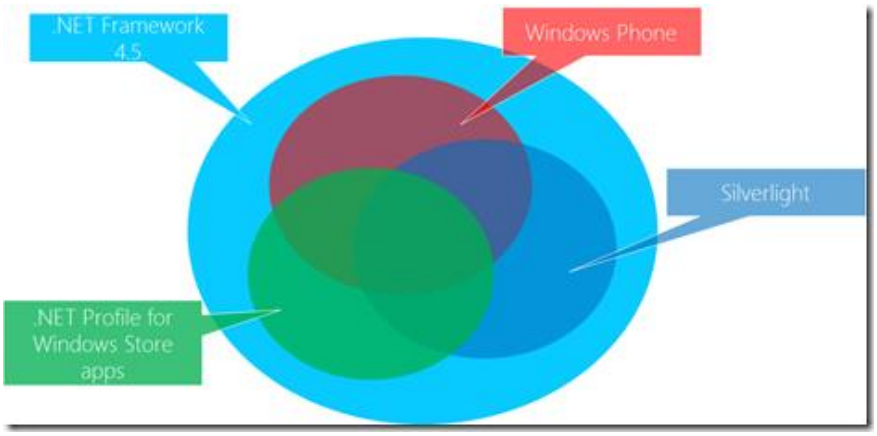
Bevor wir in die Details gehen, sollten wir uns überlegen, was technisch nicht nur machbar, sondern auch sinnvoll ist. Wo lohnt es sich tatsächlich Elemente wiederzuverwenden? Die Antwort ist recht einfach: Überall da, wo wir nicht an die Benutzeroberfläche gebunden sind. Die Benutzeroberfläche für Windows Phone und Windows 8 sieht – bedingt durch die Kacheln - zunächst sehr ähnlich aus. Bestimmte Bedienelemente erkennt man als Anwender auch sehr leicht wieder, so dass sich ein WP Anwender auf W8 recht schnell heimisch fühlt und umgekehrt. Ein Beispiel hierfür sind die Live-Tiles. Dennoch gibt es auch Unterschiede, die wir respektieren müssen. So gibt es – sicherlich nicht zuletzt bedingt durch die Bildschirmgröße - unterschiedliche UI Controls, die für die Bedienung auf den jeweiligen Geräten optimiert sind. Um viele gruppierte Datensätze anzuzeigen gibt es auf dem Windows Phone das Pivot Control. Auf Windows 8 existiert dagegen das Grid View mit der Möglichkeit Semantic Zoom zu nutzen. Wir sollten als Entwickler trotz der Unterschiede dem Anwender die optimale Bedienung erlauben. Eine Windows Phone App ist nicht einfach nur eine geschrumpfte Windows Store App. Mit diesem Wissen folgt die Erkenntnis, dass wir, was die UI Gestaltung angeht, nicht viel Spielraum haben Code 1:1 wiederzuverwenden. Es gibt ein paar Hacks, die es erlauben eigene UI Controls für beide Plattformen zu erstellen oder XAML Dateien wiederzuverwenden. Meine Empfehlung ist an dieser Stelle etwas radikaler: Tut es nicht! Versucht besser die Bedienung Eurer App für das jeweilige Endgerät zu optimieren. Es gibt ja noch andere Ecken und Enden, an denen wir durch Wiederverwendung profitieren können. Im Zuge der Benutzerschnittselle stehen wir dennoch nicht mit leeren Händen da: Unser Wissen über XAML wird uns auf beiden Plattformen weiterhelfen.

## Abkoppeln von nicht wiederverwendbaren Bereichen und MVVM

Mit diesem Wissen sollten wir versuchen nichtwiederverwendbare Teile von den wiederverwendbaren Teilen abzukoppeln. Beim UI gelingt das über Ansätze wie MVVM recht gut. Die View ist danach losgelöst von jeglicher Logik und kann – abhängig von der Zielplattform, neu implementiert und an das ViewModel gebunden werden. Auch andere Bereiche werden nicht oder nur schwer wiederverwendbar sein. So gibt es zwar ähnliche aber nicht identische Implementierungen des Process LifeCycle. Bei WP8 haben wir die Status Dormant und Tombstoned, bei W8 Suspended und Terminated. Die Konzepte sind ähnlich , es gibt aber Unterschiede. Statt hier auf Wiederverwendung von Code zu pochen, kümmern wir uns erst mal um das, was wir tatsächlich wiederverwenden können: Einen großen Teil unserer Anwendungslogik.

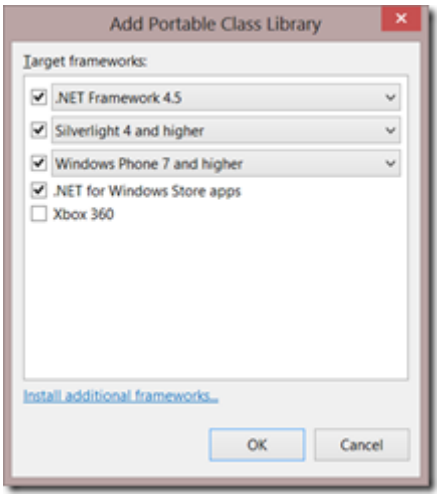
## Portable Class Library

Am angenehmsten wäre für uns als Entwickler die Arbeit, wenn wir den Logik-Code einmalig schreiben, einmalig kompilieren würden und das daraus resultierende Binary einfach auf beiden Plattformen läuft. Good news: Das funktioniert, wenn auch mit Einschränkungen. In den Visual Studio 2012 Versionen ab Visual Studio Professional aufwärts gibt es eine Projektvorlage “Portable Class Library”. Wenn wir diese auswählen, können wir Zielplattformen anwählen wie  Windows Phone 8, Windows Store Apps und Xbox. Durch das Setzen von Häkchen können wir dafür sorgen, dass der Build-Output des Projekts binär kompatibel zu allen ausgewählten Zielplattformen ist. Das klingt fast zu schön um wahr zu sein, oder? Ok, es gibt eine Einschränkung, die wir verstehen müssen. Die Portable Class Library arbeitet nach dem Prinzip des kleinsten gemeinsamen Nenners im .Net Framework. Am besten ist das zu verstehen, wenn wir uns das .Net Framework als große Menge von Namespaces und Klassen vorstellen.




Wie wir vielleicht bereits wissen, ist auf dem Windows Phone nicht das komplette .Net Framework zur Verfügung, sondern lediglich eine Submenge, die auf die Anwendung auf dem Phone zugeschnitten ist (niemand braucht ASP.NET auf dem Phone, oder? Oder vielleicht doch? Ok – wir müssen mit dem

Leben, was hier verfügbar ist.) Diese Untermenge nennt sich “Profil”. Ebenso gibt es ein Profil, das auf die Windows Store Apps zugeschnitten ist – das “.NET Profile for Windows Store Apps”. Beide Profile unterscheiden sich, sind Submengen des vollen .Net Profils, sind aber nicht deckungsgleich. Dennoch gibt es Schnittmengen. Ebenso verhält es sich mit den Profilen für die anderen auswählbaren Zielplattformen. Mit jedem Häckchen, das wir setzen reduzieren wir die Menge der zur Verfügung stehenden Namespaces auf die Schnittmenge aller Profile – und damit auf den kleinsten gemeinsamen Nenner. Die Entscheidung, welche Plattformen wir unterstützen wollen, sollten wir also sehr bewusst treffen, ein nachträgliches Aufnehmen einer weiteren Plattform wird – abhängig von der Aufgabe der Library – nur schwierig möglich sein.




## Struktur in der Solution

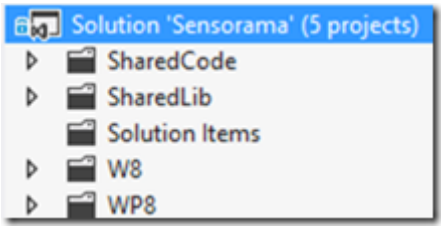
Wenn wir uns jetzt eine Solution im Visual Studio aufbauen, sollten wir darauf achten, dass die Struktur noch verständlich bleibt. Ich lege daher immer mehrere Ordner an, deren Inhalt durch die Namensgebung schon klar sein sollte:

**W8:** Ein Ordner, in dem ich später das Windows Store App Projekt ablege und weitere Projekte und Libraries, die ausschließlich für diese App relevant sind.

**WP8:** Ein Ordner, in dem ich später das Windows Phone App Projekt ablege und alle zugehörigen Projekte und Libraries, die ausschließlich für WP relevant sind.

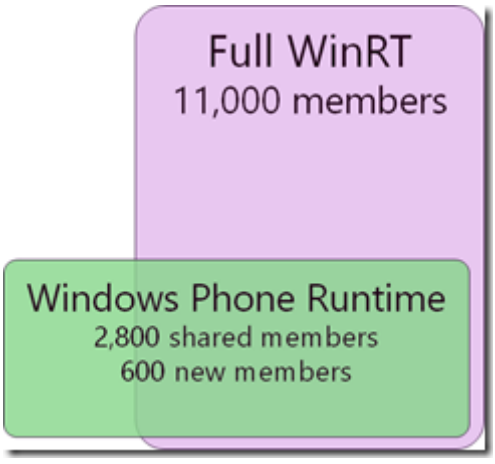
**SharedLib:** Hier lege ich meine Portable Library Projekte ab.

**SharedCode:** Hier lege ich später einzelne Code Files ab, die identisch in mehreren Projekten verwendet werden können – dazu später mehr.




Wenn wir jetzt anfangen Code zu schreiben, können wir versuchen möglichst viel in die Portable Library zu packen. Wenn ich hier alles reinpacken kann, habe ich tatsächlich sehr viel Arbeit gespart. Ich muss dann lediglich meine beiden Projekte (für WP8 und W8) anlegen und kann die Portable Libs referenzieren. Wenn ich dann noch jeweils ein eigenes UI für meine Anwendungen erstelle, bin ich fast am Ziel. Allerdings sind wir in dem, was in Portable Libraries geschehen kann  stark begrenzt, da ja, wie bereits erwähnt, mit jeder Zielplattform die zur Verfügung stehenden Namespaces weniger werden.

**Warnung:** An dieser Stelle sei eine kleine Warnung ausgesprochen: Das Vorhaben des Cross Platform Development wird sich wohl oder übel auf den Aufbau Eurer Solution und auf die Architektur auswirken. Überlegt gut, ob Ihr Eure Anwendungsarchitektur auf dem Altar des Cross Platform Development opfern wollt - und wenn ja, wie weit es sinnvoll ist, dieses Spiel zu treiben. Abhängig davon wie groß Euer Projekt ist, wie viele Entwickler Ihr seid, wie viel Erfahrung diese haben etc. müsst das zuletzt Ihr selbst entscheiden. Seid vorsichtig, lasst Euch beraten und sorgt vor allem dafür, dass Ihr selbst und Euer Team den Aufbau am Ende noch verstehen!




Spätestens, wenn wir auf die Windows (Phone) Runtime zugreifen wollen, werden wir feststellen, dass die Portable Lib meckert – wir sind ab diesem Zeitpunkt nicht mehr binär kompatibel. Da jedoch die Windows Runtime und die Windows Phone Runtime wiederum nicht absolut unterschiedlich, sondern ähnlich sind, können wir hier eventuell Code wiederverwenden – der aber zielplattformabhängig neu kompiliert werden muss. Die Namespaces der Windows Runtime und der Windows Phone Runtime

überlappen sich. Wir haben also einen Bereich von Funktionalität, deren API  auf WP8 genauso implementiert ist wie auf W8. Solange wir uns in diesem Bereich aufhalten, schreiben wir genau die gleichen Zeilen Code für W8 wie für WP8 um Windows Runtime bzw. Windows Phone Runtime Aufrufe zu machen. Was uns das bringt und inwieweit wir mit dieser Aufgabe umgehen können – dazu folgt ein Post in den nächsten Tagen. Ihr könnt ja schon mal in die Folien spitzen, was da noch so kommen mag…

## Windows 8 + Windows Phone 8 Cross Platform App Development mit C#/XAML (Teil 2)

Im letzten Teil haben wir die Luxusvariante kennengelernt, mit der wir für unterschiedliche Microsoft Plattformen nahezu ohne Mehraufwand programmieren können, die Portable Class Library. Wir haben außerdem herausgefunden, dass die Portable Class Library Einschränkungen mit sich bringt, die wir kennen und verstehen sollten und mit denen wir leben müssen. Des Weiteren kommen Entwickler, die die Express Versionen von Visual Studio nutzen leider nicht in den Genuss der Portable Class Libraries, weil diese erst aber der Professional Version zur Verfügung stehen. Grund genug also, sich anzusehen, welche Alternativen wir haben. Die Portable Class Libraries ermöglichen uns Binär-Kompatibilität. Das ist sehr angenehm für Entwickler, da der Code nur einmal kompiliert werden muss und dennoch auf allen Zielplattformen läuft, aber im Prinzip wäre uns ja auch schon geholfen, wenn wir Code wiederverwenden könnten.

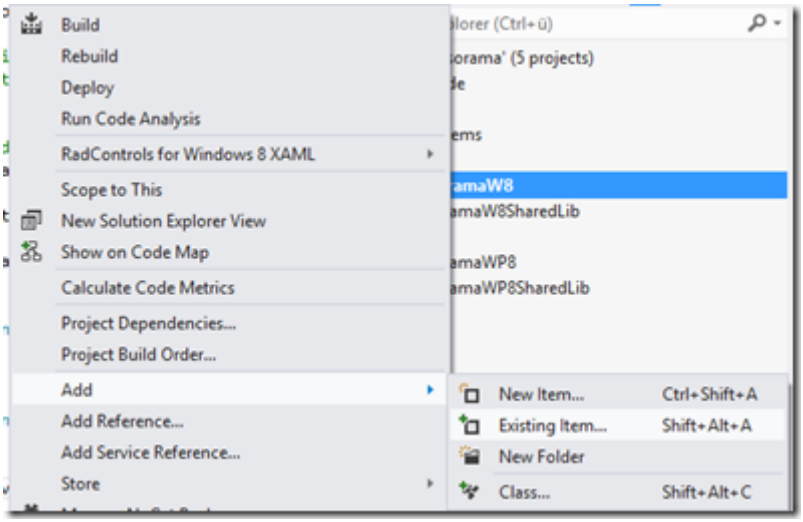
### Code Kompatibilität

Dadurch dass die Windows Phone Runtime und die Windows Runtime sich zu großen Teilen überlappen, schreiben wir die exakt gleichen Zeilen Code, wenn wir auf die Menge der APIs in diese Schnittmenge zugreifen – egal für welche der beiden Plattformen. Wir sind also “code-kompatibel” und müssen lediglich für beide Plattformen neu kompilieren. Was auf den ersten Blick erstaunlich positiv klingt, kann sich bei näherem Hinsehen leider dennoch als ziemlich umständlich erweisen: Würde das etwa bedeuten, dass wir den Quellcode aus einem Windows Phone Projekt zu gewissen Teilen per Copy & Paste in ein Windows 8 Projekt einfügen? Wäre das nicht unglaublich umständlich? Und vor allem: Wer soll das bitte pflegen? Jeder Entwickler, der schon mal Applikationen jenseits von “Hello World”- Beispielen programmiert hat, wird hier (zurecht) das Schlimmste befürchten. Ohne Unterstützung durch die Entwicklungsumgebung ist Code-Kompatibilität so attraktiv wie Nudeln ohne Soße und für professionelle Entwicklung nur sehr eingeschränkt zu gebrauchen. Man könnte natürlich als Argument anführen, dass zumindest das Wissen über die gemeinsamen APIs wiederverwendet werden kann und mit Sicherheit die Entwicklung der jeweils zweiten Plattform schneller voran gehen wird als bei der ersten. Das trifft in jedem Falle zu, der Code ist ja schon bekannt. Aber zum Glück

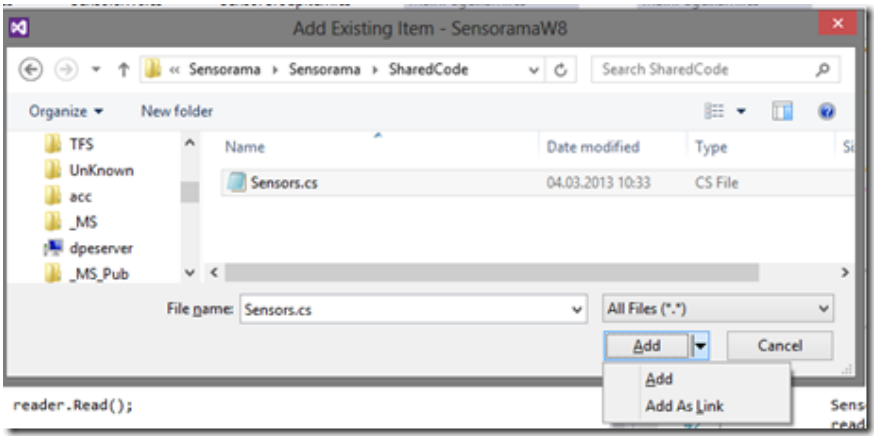
müssen wir so gar nicht argumentieren – denn wir haben ja im Visual Studio eine Toolunterstützung, wie wir sie uns wünschen. Erstaunlicherweise stelle ich immer wieder fest, dass sie nicht so bekannt ist, wie sie sein sollte.

## Referenzen auf Quellcode Dateien - Add as Link

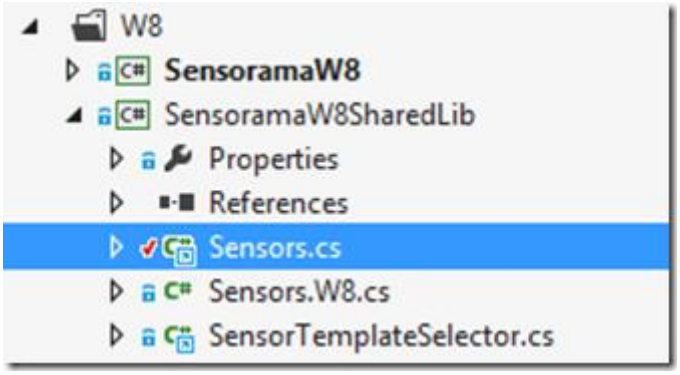
Wenn wir als Entwickler den exakt gleichen Quellcode aus einer Datei in mehreren Projekten gleichzeitig nutzen wollen, dann ist das bereits erwähnte Copy & Paste aus genannten Gründen wohl nicht die beste Wahl. Visual Studio gibt uns die Möglichkeit über den "Add" Dialog nicht nur komplett neu Dateien zu Projekten hinzuzufügen, sondern auch bestehende (Add exisiting Item). Im darauf folgenden Dialog können wir über einen Dateibrowser eine Datei anwählen.




Der Haken an der Sache ist, dass beim Hinzufügen einer bestehenden Datei aus einem anderen Projekt eine Kopie der ausgewählten Datei angelegt wird. Würden wir also eine Datei im Windows Phone 8 Projekt anlegen und anschließend aus einem Windows 8 Projekt diese Datei hinzufügen, hätten wir die Datei zweimal physikalisch auf der Festplatte liegen (einmal im jeweiligen Projektordner) und hätten lediglich eine vereinfachte Variante des rein manuellen Copy & Paste bewirkt. Was wir jedoch wollen, ist eine Referenz auf eine existierende Datei. Und siehe da: Im "Add existing Item" Dialog gibt es neben dem "Add" Button ein kleines Pfeilchen, über das wir den "Add as Link" Befehl auswählen können. Sobald wir das tun wird die Datei nicht physikalisch kopiert, sondern als Referenz eingefügt.




Für uns als Entwickler heißt das: Wenn wir einmal Code in der Datei ändern, sind automatisch beide Projekte betroffen. Wenn wir einmal neue Funktionalität einbauen, so steht diese in beiden Projekten zur Verfügung, wenn wir einmal einen Bug fixen, dann erfahren beide Projekte diesen Fix. Im Umkehrschluss müssen wir auch beide Projekte testen, sobald wir die Datei modifiziert haben. Dateien, die nicht als physikalische Datei hinzugefügt wurden, sondern lediglich als Referenz, erkennen wir am kleinen blauen Pfeilchen im Logo im Solution Explorer. So sehen wir auf einen Blick, ob wir die Datei physikalisch oder als Referenz vorliegen haben. Eine solche referenzierte Datei kann – logischerweise – auch nur einmal im Visual Studio geöffnet sein. Würde man versuchen die Datei neu aus einem anderen Projekt der gleichen Visual Studio Instanz zu öffnen, so meldet uns Visual Studio, dass das nicht möglich ist, da sie schon im Rahmen eines anderen Projektes geöffnet ist.




Wann immer wir die Windows Runtime im Bereich der Schnittmenge zwischen Windows Phone Runtime und Windows Runtime ansprechen, können wir hier von gleichem Code und diesem Feature profitieren und den Code für mehr als nur ein Projekt nutzen. Ich persönlich lege diese gemeinsam genutzten Dateien in einem separaten Solution Folder ab, den ich bereits im letzten Posting angekündigt habe: Der SharedCode Folder. Damit ist klar, dass die Datei – egal aus welchem Projekt ich diese letztlich referenziere – nicht einem Projekt exklusiv gehört.

## Partial Classes

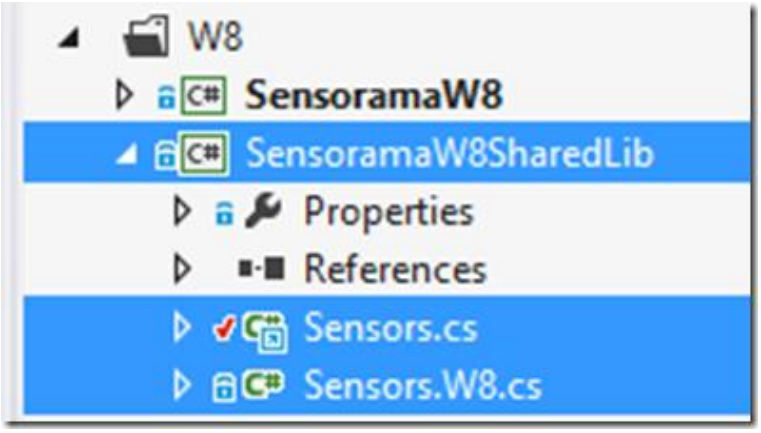
Die beiden bisherigen Strategien **Shared Code über Referenzen** und **Portable Class Libraries** eignen sich sehr gut, um ViewModels im MVVM-Aufbau abzubilden. Allerdings werden wir bei der Verwendung auch hier früher oder später an die Grenzen des Trivialen stoßen. Vermutlich werden wir in unseren Projekten auch plattformspezifische Bereiche der Runtime nutzen. Beispielsweise Windows 8 Runtime APIs, die es auf dem Phone nicht gibt und umgekehrt. Dadurch, dass in der Regel eine Datei in C# eine Klasse darstellt und wir durch die Add-As-Link-Strategie immer vollständige Dateien referenzieren, schränkt uns die Anwendung dieser Strategie de facto darauf ein innerhalb einer Klasse, die wir teilen möchten, ausschließlich in der Schnittmenge der Windows Runtime zu operieren. Wenn wir hier flexibler sein möchten, bietet sich aber eine sehr einfache Lösung an: Das Verwenden von Partial Classes.

Das Feature der Partial Classes ist schon seit einigen .NET Versionen vertreten und letztlich gibt es uns die Möglichkeit den Inhalt einer Klasse über mehrere Dateien in Einzelteile (Parts) zu zerlegen. Durch das Keyword "partial" markieren wir eine Klasse als Partial Class. Der Compiler erkennt dann, dass er vor dem Kompilieren nochmal nach links und rechts blicken muss, um nach weiteren Parts zu suchen. Wenn er welche findet, dann bindet er diese mit in den Kompiliervorgang mit ein. Wenn nicht, dann eben nicht.

Wenn wir nun also eine Klasse geschrieben haben, in der beispielsweise 9 von 10 der Methoden Code-Kompatibilität aufweisen und lediglich eine aus dem Ruder läuft, da hier eine plattformspezifische API angesprochen wird, bietet es sich an diese eine Methode in einen separaten Part zu schieben und somit vom Rest des Codes zu trennen. An dieser Stelle sollten wir uns auf eine Namenskonvention einigen, da sonst schnell der Punkt erreicht ist, an dem niemand mehr versteht, was wohin und wozu gehört. Die Namenskonvention übernehmen wir von den XAML Dateien: Wenn wir uns diese genau ansehen, werden wir feststellen, dass es zusätzlich zu den *Dateiname.xaml* Dateien auch noch eine *Dateiname.xaml.cs* Datei finden. Der naheliegende Vorschlag ist also den Dateinamen beizubehalten und über einen zusätzlichen Namenszusatz zu markieren: Dateiname.W8.cs für Windows 8 spezifischen Code, Dateiname.WP8.cs für Windows Phone 8 spezifischen Code.
Um die Struktur einfach zu halten, wäre mein Vorschlag, dass der gemeinsame Teil der Klasse wieder im SharedCode Folder abgelegt wird. Von dort wird er in den jeweiligen Projekten referenziert. Die plattformspezifischen Klassenteile werden im jeweiligen Projekt abgelegt – sind also nur dort zu finden, wo sie wirklich von Bedeutung sind.
Das Ganze könnte im Code dann wie folgt aussehen:

```
12      partial class SensorReader
13      {
14          private void RegisterLightSensor()
15          {
16              myLightSensor = LightSensor.GetDefault();
17              if (myLightSensor != null)
18              {
19                  myLightSensor.ReportInterval = myLightSen
20                  myLightSensor.ReadingChanged += new Typed
21              }
22
23          }
```

Und im Visual Studio Solution Explorer so (in diesem Fall habe ich die beiden Klassenteile Sensors.cs und Sensors.W8.cs nicht direkt ins Windows 8 Projekt gehängt, sondern eine weitere Library angelegt, die alle Shared Anteile beinhaltet und diese wiederum aus meinem eigentlichen Windows Store App Projekt referenziert. Nicht verwirren lassen!):




Unterm Strich bleibt stehen: Durch Partial Classes erhöhen wir die Granularität der wiederverwendbaren Häppchen. Plattformspezifische Besonderheiten können wir durch weitere Teilklassen hinzufügen. Unsere wiederverwendbaren Teile entsprechen also nicht mehr zwingend einer ganzen Klasse entsprechen, sondern wir können auch nur mit Teilbereichen von Klassen arbeiten.

Auch an dieser Stelle scheint mir eine **Warnung** sinnvoll: Wie Ihr seht, erhöhen wir hier Schritt für Schritt die Wiederverwendbarkeit einzelner Assets innerhalb unserer Projektmappe. Das ist ja auch Ziel dieses Postings. Bitte beachtet, dass im gleichen Maße die Komplexität steigt, der Ihr als Entwicklerteam gegenübersteht.  Was im Beispiel mit "Hello World"-Niveau ganz locker geht, wird beim Entwickeln im 100-Mann Team vielleicht nicht ganz so reibungslos verlaufen. Vielleicht hat sich auch jemand den oberen Abschnitt schon 2 mal durchgelesen um ihn zu verstehen? Falls ja: Geht davon aus, dass das Eure Kollegen auch tun müssen. Dennoch sind Partial Classes eine feine Sache – überlegt Euch einfach, wo Ihr im Verhältnis Aufwandsersparnis vs. Komplexität steht und ob diese

Strategien für Euren Anwendungsfall eine valide Möglichkeit darstellt. Erklärt den neuen Mitgliedern in Eurem Team, wie der Quellcode aufgebaut ist. Partial Classes nutzt in dieser Verwendung sicher nicht jeder.

## Bedingte Kompilierung

Nachdem wir uns jetzt ausgehend von physikalischen Dateien über Klassen auf Klassenteile (und damit nicht zuletzt einer bestimmten Menge von Methoden)heruntergehangelt haben, können wir noch einen Schritt weiter gehen: Wie wär's denn, wenn wir einzelne Codezeilen wiederverwenden könnten? Auch das kriegen wir im Visual Studio über **Bedingte Kompilierung** hin. Bedingte Kompilierung funktioniert über kleine Code Schnipsel, mit denen wir den Compiler dazu überreden können bestimmte Codeabschnitte wahlweise einzubinden oder zu ignorieren. In dem Szenario, in dem wir uns befinden bedeutet das, dass wir eine Datei über "Add-As-Link" in unsere beiden Zielprojekte einbinden. Über Anweisungen im Code können wir dann bestimmte Aufrufe, die nur für eine der beiden Plattformen gilt ein und ausblenden.

Diese Aufrufe sind nichts anderes als eine if-Bedingung mit vorangestelltem #, das die Zielplattform abfragt. Wenn die gleiche Datei einmal in ein W8 und einmal in ein WP8 Projekt eingebunden wird, wird abhängig davon welches Projekt gerade angewählt ist der jeweils ungültige Quellcode ausgegraut und der andere normal dargestellt. Im Prinzip könnten wir über diesen Mechanismus das Verfahren mit den Partial Classes ersetzen.

```
#if NETFX_CORE
            SensorRegisterHelper(RegisterLightSensor);
#endif
```

Allerdings lautet meine Empfehlungen hier anders: Ich bin zwar nicht der Meinung, dass Bedingte Kompilierung generell böse ist, aber sie sorgt dafür, dass der Quellcode in kürzester Zeit sehr sehr (diese beiden "sehr" sind durchaus als Steigerung zu sehen) undurchschaubar wird. Als Entwickler bekommt man durch die Farbgebung der IDE zwar eine gewisse Unterstützung um herauszufinden, welcher Code gerade relevant ist. Dennoch sehen wir als Entwickler ständig Codeabschnitte ohne Bedeutung für unser aktuelles Projekt und arbeiten mit mehr Zeilen Code als wir benötigen. Für alle Codeabschnitte, die über sehr wenige Zeilen hinausgehen, würde ich das Verwenden von Compiler Anweisungen nur unter Vorbehalt empfehlen und stattdessen zu Partial Classes tendieren, falls sich das umsetzen lässt.

Für using-Statements am Dateianfang hingegen eignen sie sich sehr gut – falls also die gleiche Funktion auf W8 und WP8 in unterschiedlichen Namespaces zu finden ist, so wäre das ein sehr gutes Einsatzgebiet der Compiler Anweisungen. An dieser Stelle spare ich mir die Warnung. Ich denke, ich bin bereits in der Erläuterung genug auf die Nachteile der Compiler-Anweisungen eingegangen.

Dieser Post zeigt, dass wir unter Verwendung von .NET ein paar schöne Möglichkeiten haben unseren Quellcodeaufbau so zu gestalten, dass er wiederverwendbar wird. Nichts davon ist wirklich neu für Windows 8 oder Windows Phone 8 Apps – wir müssen lediglich den Aufbau unserer Lösung auf den geplanten Anwendungsfall hin optimieren. So kommen wir mit ein bisschen Basis-Handwerkszeug unserem eigentlichen Ziel noch ein bisschen näher. Natürlich lassen sich die hier beschriebenen Strategien hervorragend mit der Verwendung von Portable Class Libraries kombinieren. Und dann könnten wir uns ja neben dem physikalischen Aufbau unserer Solution auch noch mit dem logischen Aufbau beschäftigen … aber dafür gibt's einen neuen Post.

## Windows 8 + Windows Phone 8 Cross Platform App Development mit C#/XAML (Teil 3)

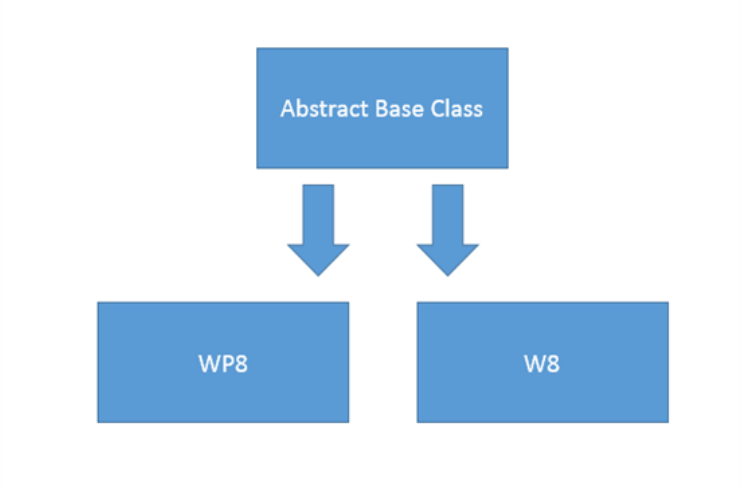
Im ersten Teil dieser Serie haben wir einen Blick darauf geworfen, welche Möglichkeiten sich uns überhaupt bieten Code zwischen Windows 8 und Windows Phone 8 wiederzuverwenden. Wir haben die Portable Libraries kennengelernt und deren Einschränkungen erfahren. Im zweiten Teil haben wir uns unterschiedliche Methoden angesehen, wie wir durch geschickte Dateiverwaltung und ein paar Visual Studio Features Quellcode Artefakte zwischen mehreren Projekten teilen können. Natürlich können wir der Aufgabe plattformübergreifen Code zu entwickeln auch durch die Komponenten-Design Brille begegnen.

### Vererbung

Wenn wir für zwei oder mehr Plattformen gleichzeitig entwickeln wollen (oder sollen), können wir diese Aufgaben ja auch wie folgt formulieren:
*Wir entwickeln gleiche oder für mehrere Plattformen, wobei manche der Funktionen sich in der Implementierung unterscheiden (also plattformspezifisch sind), ein Teil der Funktionen aber gleich ist.*

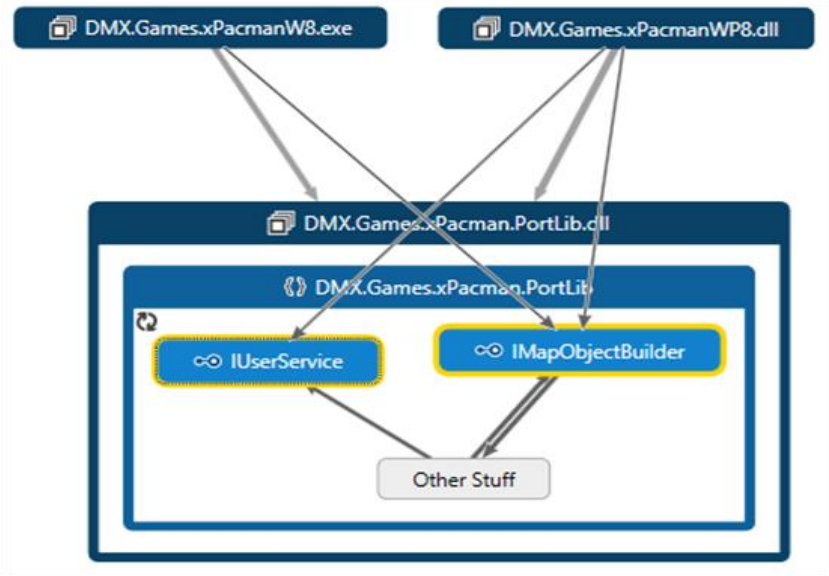
Bei Entwicklern von objektorientierten Sprachen klingelt' s eventuell schon: Natürlich ist das ein Vererbungspräzedenzfall. Wir könnten also die gemeinsame Funktionalität einfach in Basisklassen auslagern und die plattformspezifischen Implementierungen von einzelnen Funktionen zusätzlich in abgeleiteten Klassen zur Verfügung stellen.
Ein Vorteil wäre die gemeinsame Basis, die nur einmal geschrieben und getestet werden muss. Ein

Nachteil natürlich die Vererbungskette, die hier um eine Stufe wächst. Natürlich muss unser Klassendesign insgesamt stimmig sein – die Basisklasse können wir dann beispielsweise als Portable Class Library verpacken oder über „Add as Link“ als Referenz zum jeweiligen Projekt hinzufügen.




## Interfaces

Alternativ oder erweiternd dazu bietet sich das Arbeiten mit Interfaces an. Wie wir bereits erfahren haben, gibt es typischerweise einen Bereich des Codes, der plattformübergreifend gültig ist. Einzelne Bereiche sind aber plattformspezifisch zu implementieren. Wenn wir nun unsere wiederverwendbaren Klassen so designen, dass diese nicht mit konkreten Implementierungen arbeiten, sondern lediglich mit Interfaces, so können wir die Implementierungen dieser Interfaces in einer plattformspezifischen Klasse erledigen. Die konkrete plattformspezifische Implementierung wird dann bspw. im Konstruktor der plattformunabhängigen Komponente übergeben. Unterm Strich eine Form von Dependency Injection. Das Arbeiten mit Interfaces hat den Vorteil, dass wir uns ausgiebig mit dem Aufbau unserer Solution beschäftigen müssen und gegebenenfalls eine sehr modulare Lösung erzeugen. Der Nachteil liegt in einer - meinem Empfinden nach - etwas höheren Komplexität.



## Delegates

Bisweilen funktioniert der Mechanismus, den ich über die Interfaces erreichen kann auch mit Delegates. In diesem Falle ruft mein plattformunabhängiger Code kein Interface auf, sondern eben einen Delegate. Das ist auf den ersten Blick ein bisschen leichtgewichtiger, als die Definition eines Interfaces. Persönlich empfinde ich das Arbeiten mit Interfaces aber als sauberer, da man die Interfaces bereits „von außen" erkennt, beispielsweise im Architekturtool des Visual Studio (siehe auch oben), während man die Delgaten erst im Code entdeckt. Statt ein Inteface zu implementieren könnten wir so einfach eine Methode übergeben, die unter bestimmten Umständen aufgerufen wird.

## Up-/Down-Casting

Um meinen Blog Post hier auch mit praktischer Erfahrung zu belegen, habe ich ein paar kleinere Projekte für Windows 8 und Windows Phone 8 entwickelt, mit dem Vorsatz diese so zu strukturieren, dass ich möglichst wenig plattformspezifischen Code schreiben muss. Ich habe dabei alle hier genannten Mechanismen genutzt, will aber nicht verhehlen, dass ich auch noch ein bisschen mehr verwenden musste.
Als etwas unsauber empfand ich dabei das Up-/Downcasting auf den Typen „Object". In meiner Portable Class Library habe ich Referenzen auf Objekte gespeichert, die – abhängig davon auf welcher Plattform die Lib zum Einsatz kam – unterschiedlichen Typs waren. Ich musste hier mit einer Referenz arbeiten – der einfachste Weg schien mir das Hinterlegen als „Object" zu sein. In der jeweiligen Plattform musste ich dann bei Verwendung dieser Referenz entsprechend casten.  Das konnte ich natürlich problemlos tun, da ich ja auf der jeweiligen Plattform weiß, welchen Typ ich erwarte. Richtig sauber fühlt sich das leider nicht an, aber es schien mir eine pragmatische Lösung zu sein. Natürlich empfiehlt es sich hier ein wenig über Exception Handling nachzudenken…

```
public abstract class MyAbstractObject
{
    // ...

    // aaargh! I don't know the type. And even if I did...
    public object myRefObject { get; set; }

    // ...
}
```

## Fazit

Damit bin ich vorerst am Ende meiner Serie über Cross Platform Development zwischen Windows Phone und Windows 8. Die hier genannten Vorgehensweisen könnt Ihr aber ebenso gut auch bei Entwicklung von Serverkomponenten, Desktopsoftware oder sonstiger Windows-Entwicklung

heranziehen. Natürlich ist Tauglichkeit und Nutzen immer abhängig vom konkreten Szenario, aber es schadet sicher nicht, sie alle in einem Blogpost gelistet zu finden.

Was ist jetzt mein Fazit? Mein persönliches Fazit ist deutlich besser, als ich erwartet hatte, bevor ich mich mit der Materie befasst habe. Plattformübergreifende Entwicklung mit C# für Windows 8 und Windows Phone 8 ist möglich – wenn man sich mit den Rahmenparametern beschäftigt. Das umfasst unterschiedliche .NET Profile, Unterschiede in der Windows Runtime und im UI sowie im Laufzeitverhalten von Applikationen. Die Logik der Anwendung jedoch ist weitgehend unbeeinflusst – und hier kann man mit den genannten Vorgehensweisen punkten.

Ist also alles super? Fairerweise muss ich auch hier etwas bremsen: Die Plattformunabhängigkeit bekommt man nicht umsonst. Es wird in jedem Falle etwas Zeit kosten unsere Lösung plattformunabhängig zu gestalten. Ein paar der hier aufgezeigten Vorgehensweisen lohnen aber ohnehin einer näheren Betrachtung, da sie ein paar nette Nebeneffekte Mitbringen – Modularität zum Beispiel. Insofern lohnt sich das Investment vielleicht so oder so.

## Peter Kuhn (AIT GmbH) – Nicht nur mobil: die neuen Windows Azure Mobile Services näher beleuchtet

### Arbeitgeber

• AIT GmbH & Co. KG

### Technologieschwerpunkte

• .NET

• Windows Phone

• Softwaredesign und –architektur

• Code Reviews

### Mein Urlaubstipp

Japan – diesen wohlig-wundersamen Kulturschock sollte sich niemand entgehen lassen.

### Meine Inseltechnologie

T4-Templates haben es im vergangen Jahr in Rockstar-Manier auf einen Spitzenplatz in meinem Repertoire geschafft. Je länger man sich damit beschäftigt und je mehr man sich auf das Thema Metaprogrammierung generell einlässt, desto häufiger stolpert man über Einsatzfelder, und desto mehr lernt man die Konzepte und Vorteile zu schätzen – einmal damit angefangen gibt es kein Zurück mehr. Faszinierende Technologie und Möglichkeiten!

### Meine Sommerlektüre

Urban Priol: Hirn ist aus – um sich mit schnoddrig-böser Satire vom Stress des Alltags zu befreien.

## Nicht nur mobil: die neuen Windows Azure Mobile Services näher beleuchtet

Ein weiterer Sprössling im Reigen der Features von Windows Azure soll es Entwicklern deutlich erleichtern, Cloud-Backends für typische Szenarien mobiler Clients zu erstellen.

Seit Scott Guthrie das Ruder übernommen hat, erfahren nicht nur die Entwicklerwerkzeuge für Windows Azure kontinuierlich Verbesserungen. Auch neue Features der Plattform selbst scheinen in letzter Zeit in immer kürzeren Abständen angekündigt zu werden. Die Windows Azure Mobile Services als eine der jüngsten Neuerungen erleichtern die Entwicklung von Cloud-basierten Backends, beispielsweise für Windows-8-Apps.

Als Entwickler benötigt man zunächst einen Account auf www.windowsazure.com, den man als kostenlose Trialversion einrichten kann. MSDN-Abonnenten hingegen können direkt über die Subscription Benefits ihres Abonnements einen solchen Account erstellen und kommen so in den Genuss einiger zusätzlicher kostenfreier Inklusivleistungen [1]. Nachdem man in seinem Azure-Account im Bereich Preview Features die Mobile Services freigeschaltet hat, benötigt man lediglich noch eine Reihe von Werkzeugen, um mit der Entwicklung loslegen zu können, hier beispielhaft für das Microsoft-Umfeld aufgeführt:

- Visual Studio 2012, zum Beispiel in der Express Edition [2],
- das Mobile Services SDK [3],
- das Live SDK (nur für Authentifizierung und Autorisierung über Live.com) [4].

Die primär unterstützten Clientplattformen sind momentan WinRT (Windows 8 Store Apps), Windows Phone 8 und iOS. Die Unterstützung weiterer Plattformen ist bereits angekündigt oder von Drittherstellern verfügbar, zum Bei- spiel gibt es durch eine Partnerschaft Microsofts mit Xamarin bereits jetzt Libraries für Android [5]. Da der technische Unterbau auf REST-Endpoints mit einem JSON/ODATA-Format setzt, dürfte die Anbindung weiterer Plattformen unter Zuhilfenahme der Dokumentation [6] aber auch ohne explizite Unterstützung relativ einfach zu machen sein. Lesen Sie dazu auch den Kasten Windows Azure Preview Features.

**Windows Azure Preview Features**

Wie einige andere neu hinzugekommene Funktionen der letzten Zeit werden auch die Mobile Services zunächst als sogenanntes Preview Feature zugänglich gemacht. Auch wenn damit ein schöner neuer Name für Betaversionen gefunden wurde, bleibt der Kern der Sache doch gleich: Preview Features können noch Fehler enthalten und stellen eventuell nicht den vollen Funktionsumfang bereit, was man immer im Hinterkopf behalten sollte. Bei der Nutzung erkennt man diesen Vorabcharakter mit den Lizenzbedingungen an und akzeptiert, dass man sich außerhalb des gewöhnlichen Service Level Agreements für Windows Azure bewegt [7]. Im Gegenzug werden Preview Features meistens verbilligt oder gar kostenlos angeboten. Von den Mobile Services darf man zehn Instanzen für zwölf Monate kostenlos betreiben [8]. Eine Einschränkung dabei ist, dass nur 165 MByte ausgehender Datenverkehr pro Tag enthalten sind.

## Erste Schritte

Mit dem Versprechen, innerhalb weniger Minuten eine Anwendung für Windows 8 inklusive Datenbank-Backend in der Cloud zu erstellen, hat Microsoft keineswegs übertrieben. Wer sich erst einmal orientiert und den Umgang mit der Verwaltungsoberfläche verinnerlicht hat, kann tatsächlich innerhalb kürzester Zeit ein voll funktionsfähiges Grundgerüst für eine App aufbauen. Beim Anlegen von Mobile Services legt man zunächst den URL fest, unter welchem diese erreichbar sein sollen (hier: braindump. azure-mobile.net), und wählt die SQL-Azure-Datenbank aus, die als Speicher verwendet wer- den soll (siehe Abbildung 1).

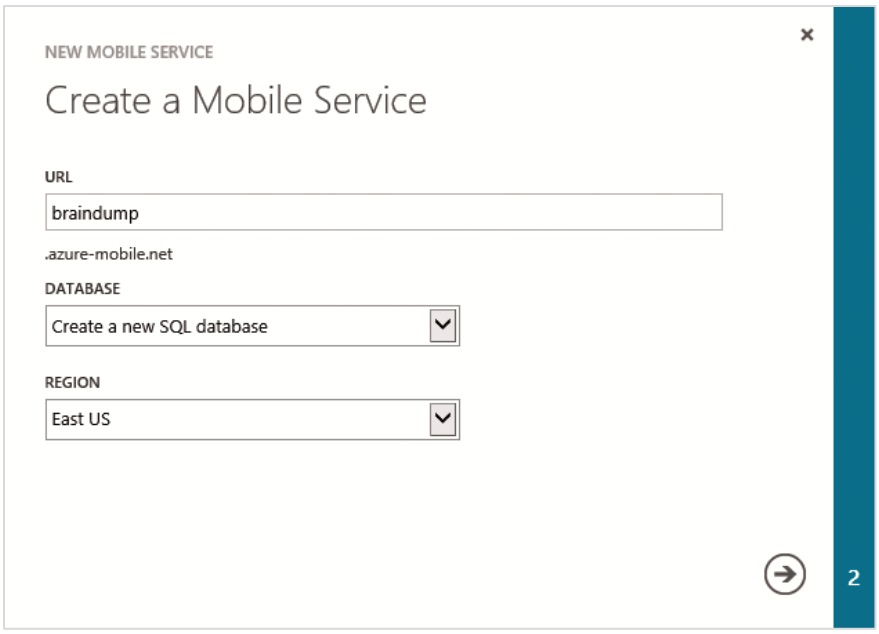



*Abbildung 1*

Wer noch keine Datenbank erstellt hat, kann dies direkt über den Assistenten erledigen (siehe Abbildung 2). Die Datenbank sollte in derselben Region gehostet werden wie die Mobile Services, um unnötigen Traffic zu vermeiden. Scott Guthrie hat übrigens bereits angekündigt, dass man in einem späteren Release neben SQL Azure auch weitere Speicherformate unterstützen wird. Genauere Informationen hierüber liegen aber noch nicht vor.

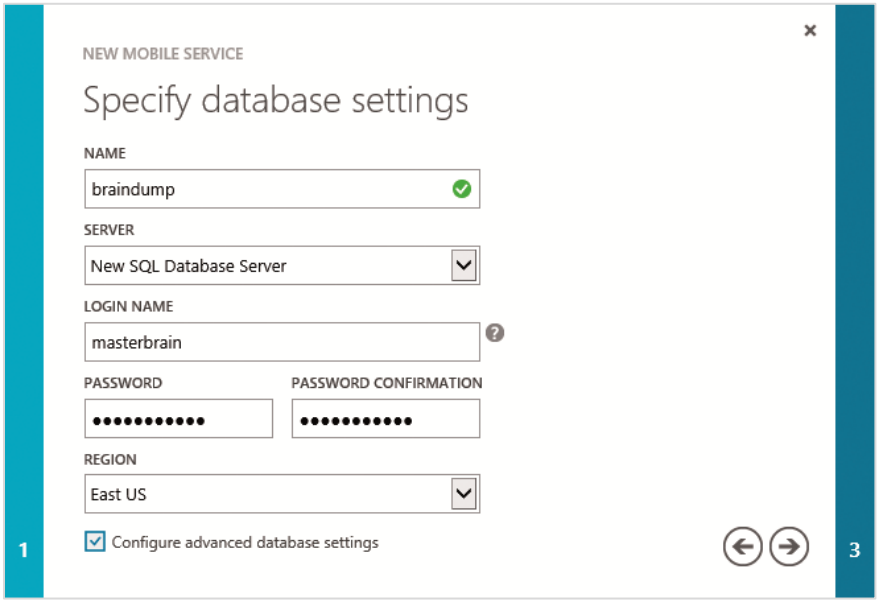



*Abbildung 2*

Die Weboberfläche bietet nun an, mit wenigen Klicks eine voll funktionsfähige Beispiel- App zur Verwaltung von Aufgaben (Todos) zusammenzubauen – von der benötigten Datenbanktabelle bis zur

fertigen Solution für Visual Studio, wahlweise in C# oder JavaScript (JavaScript wird nur unter Windows 8, nicht unter Windows Phone 8 unterstützt). Für iOS steht ein vorkonfiguriertes Projekt für Xcode zur Verfügung. Dies ist ein idealer Einstiegspunkt, um ein überschaubares Beispiel explorativ kennenzulernen.

Möchte man stattdessen seine Anwendung manuell konfigurieren, gilt es zunächst einmal mindestens eine Tabelle anzulegen, was man über den Data-Reiter seines eben erzeugten Mobile Service erledigt. Neben dem Namen der Tabelle (hier: Thoughts) wird zunächst lediglich abgefragt, welche Berechtigungen benötigt werden, um die üblichen CRUD-Operationen durchzuführen (siehe Abbildung 3).

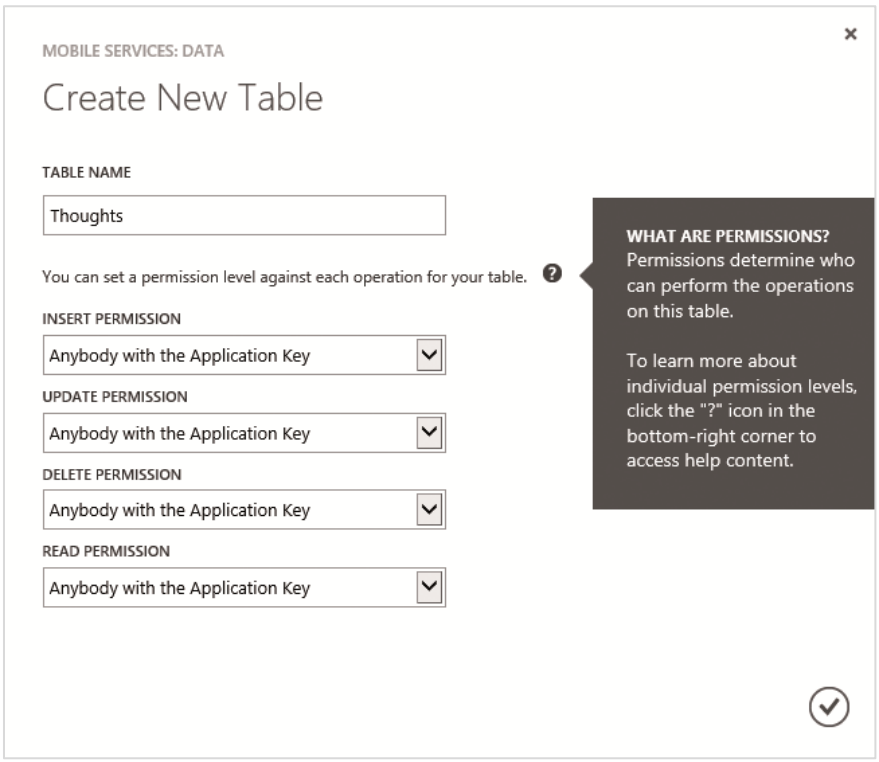



*Abbildung 3*

Nach wenigen Augenblicken taucht die neu angelegte Tabelle in der Liste auf, und man kann in der Detailansicht die Daten, Berechtigungen und Spalten erneut ansehen. Irritierend ist aber, dass man zur einzigen bereits existierenden, automatisch angelegten id-Spalte hier keine weiteren Spalten hinzufügen kann. Stattdessen greift ein Feature namens Dynamic Schema, das man in den Konfigurationseinstellungen seines Mobile Service an- und abschalten kann (siehe Abbildung 4). Es sorgt dafür, dass bei eingehenden Daten automatisch geprüft wird, ob das Schema geeignet ist, diese Daten zu speichern. Gegebenenfalls werden dynamisch einfach neue Spalten mit den passenden Datentypen hinzugefügt. Standardmäßig ist Dynamic Schema zunächst aktiviert; aus verständlichen Gründen sollte man die Funktion vor der Auslieferung seiner App abschalten.

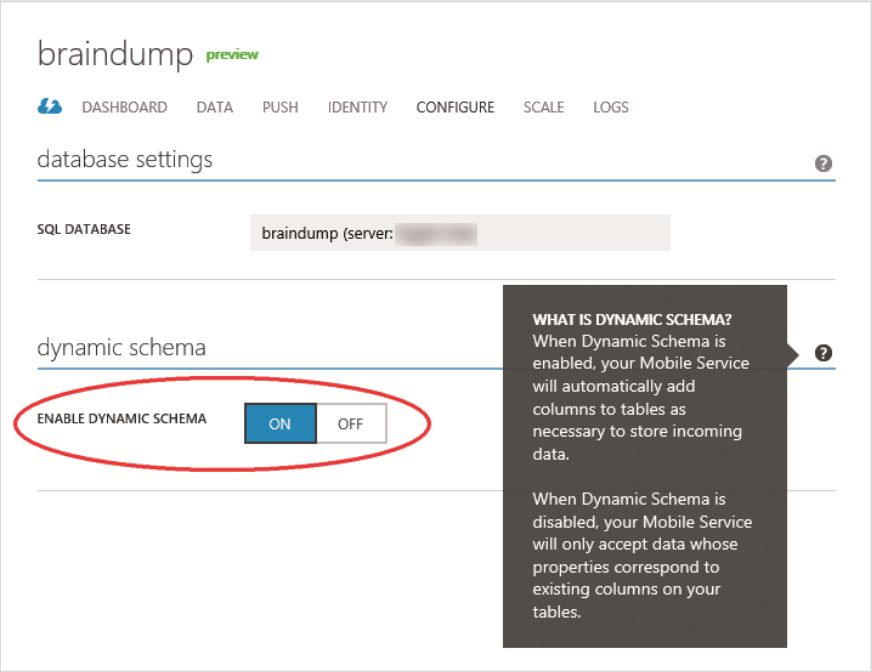



*Abbildung 4*

Selbstverständlich kann man auch die gewöhnlichen Tools von Windows Azure verwenden, um die Datenbank zu verwalten, oder dies direkt mit seinen gewohnten lokalen Datenbankwerkzeugen erledigen. Letztendlich handelt es sich bei den Werkzeugen der Mobile Services nur um einen vereinfachten Zugriff auf die dahinterliegende SQL-Azure-Datenbank.

## Anbinden eines Clients

Für einen möglichst reibungslosen Start bei der Cliententwicklung bietet die Einstiegsseite der Mobile Services kleine Hilfestellungen an und liefert zudem Code-Snippets für alle unterstützten Plattformen, die bereits die korrekten Daten wie die vorkonfigurierten Werte für Service-URL und Application Key enthalten (siehe Abbildung 5).

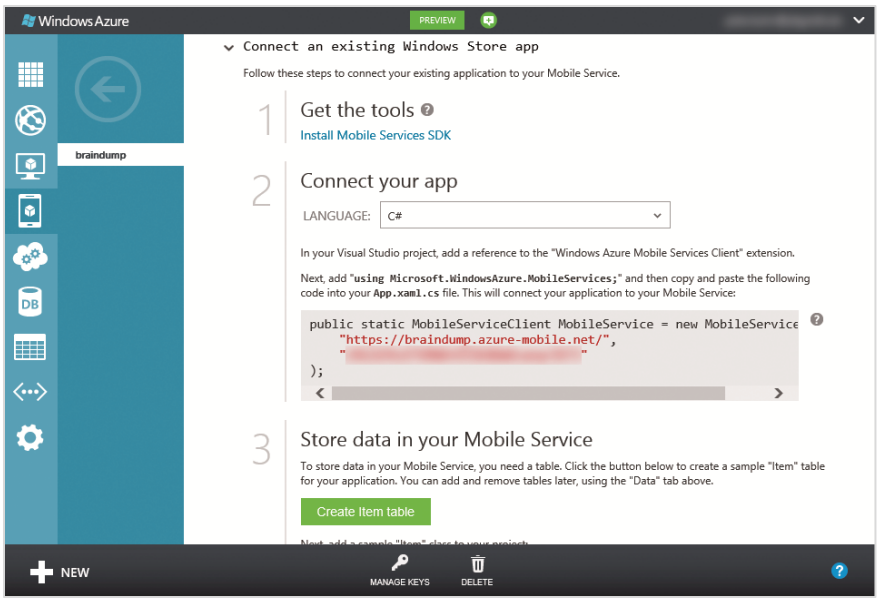



*Abbildung 5*

Zunächst gilt es, seine Entitäten nach dem Vorbild der erstellten Tabellen im Client zu modellieren. Hierzu legt man gewöhnliche Klassen an, die man gegebenenfalls mittels der DataTable- und DataMember-Attribute auf die korrekten Tabellen und Spalten seines Mobile Service abbildet. Das ist hilfreich, um mit Unterschieden bei Groß-/Kleinschreibung, im Numerus oder mit allgemein abweichen- den Begrifflichkeiten umzugehen.

```
[DataTable(Name = "Thoughts")]
public class Thought
{
        [DataMember(Name = "id")]
        public int Id { get; set; }

        public string Title { get; set; }
        public string Text { get; set; }
}
```

Zur Kontaktaufnahme mit dem Service wird eine Instanz der Klasse MobileServiceClient benötigt. In seinen Codebeispielen verwendet Microsoft ein Pseudo- Singleton, das in der App-Klasse platziert ist und dadurch im gesamten Code verfügbar gemacht wird.

```
public static MobileServiceClient
        MobileService = new MobileServiceClient(
                "https://braindump.azure-mobile.net/",
                "[Application Key]"
);
```

Die Clientinstanz erlaubt den Zugriff auf Tabellen zu einem bestimmten Typ, die wiederum Methoden für die üblichen CRUD-Operationen anbieten. Das Einfügen eines neuen Elements ist hierdurch mit wenigen Zeilen Code erledigt:

```
// create new item
var thought = new Thought
{
        Title = TitleInput.Text,
        Text = TextInput.Text
        };

// get table and invoke async insert
var thoughtsTable =
        App.MobileService.GetTable<Thought>();
        await thoughtsTable.InsertAsync(thought);

// the Id property now contains the
// auto-incremented value
Debug.WriteLine(thought.Id);
```

Die angebotenen APIs sind, wie bei Windows 8/WinRT üblich, asynchron ausgeführt, sofern sie Netzwerkoperationen auslösen. Nach einem Probelauf der Anwendung und erfolgreichem Einfügen eines neuen Datensatzes wurden von den Mobile Services durch die „Magie“ von Dynamic Schema die fehlenden Spalten in der Tabelle eigenständig eingefügt. Die Typen wurden automatisch aus den empfangenen Daten abgeleitet (hier: String, siehe Abbildung 6).

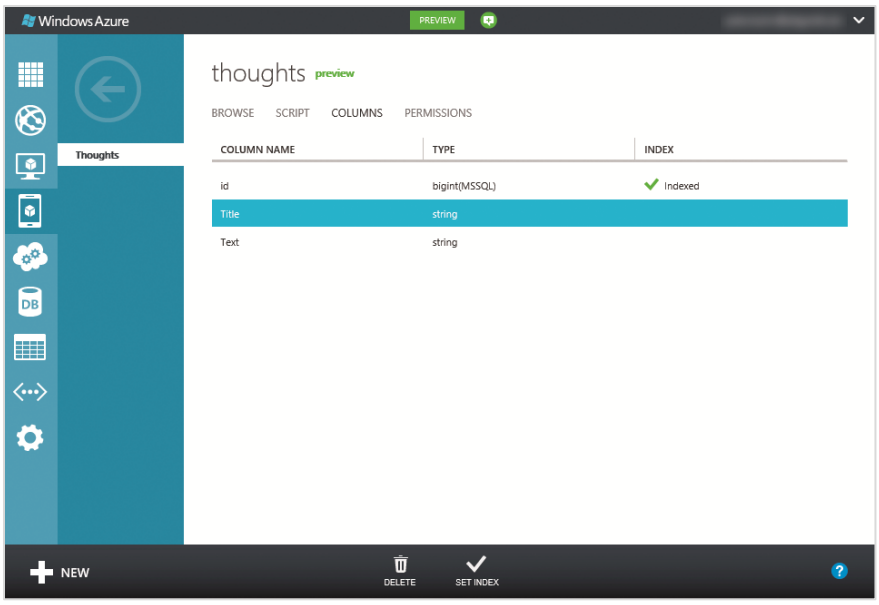



*Abbildung 6*

Beim Abfragen von Daten gibt es neben den zu erwartenden Methoden fürs Selektieren mit Unterstützung für Filtern, Sortieren und Paging auch einige Convenience-Methoden, welche die direkte Anbindung an entsprechende UI-Elemente erlauben, beispielsweise die Konvertierung zu einer CollectionView:

```
var thoughtsTable =
        App.MobileService.GetTable<Thought>();
var thoughts =
        thoughtsTable.ToCollectionView();

ListItems.ItemsSource = thoughts;
```

Hinter den Kulissen erfolgt natürlich der Abruf der Daten wiederum asynchron, und die Darstellung in der Obe fläche wird über die üblichen Binding- Mechanismen aktualisiert, sobald die Daten eintrudeln.

## Business Rules

Ein Problemkreis bei derart generischen Backend-Frameworks ist immer, genügend Erweiterungsmöglichkeiten zu schaffen, um die Abläufe an die eigenen Bedürfnisse anpassen zu können. Beispielsweise möchte man fast immer aus Sicherheits- und Konsistenzgründen serverseitig zusätzliche Logik implementieren, die unabhängig vom verwendeten Client ausgeführt wird.

Zu diesem Zweck wurden serverseitige Skripte als Extension Points geschaffen, in die man seinen eigenen Code einklinken kann, wiederum jeweils bei den üblichen CRUD-Operationen. Das Beispiel zeigt, wie man beim Insert ein Feld des Items (ChangedAt) auf das aktuelle Datum setzt (siehe Abbildung 7).

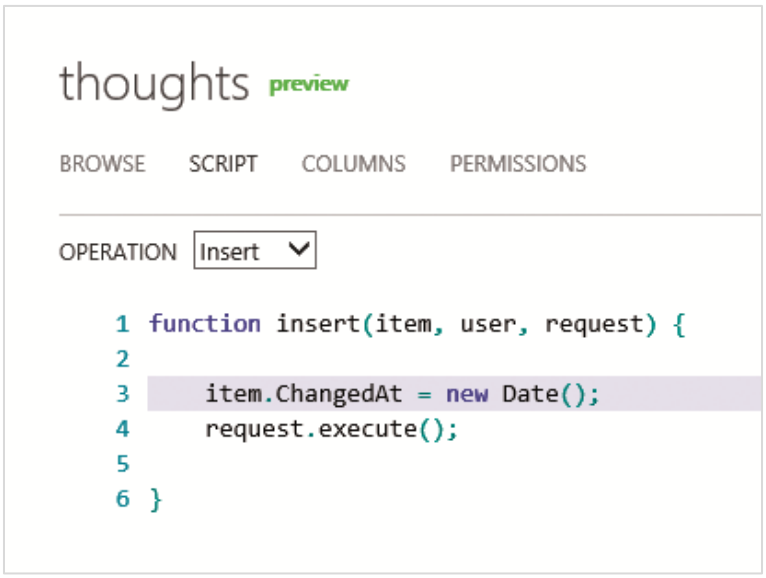



*Abbildung 7*

Sofort fällt auf, dass es sich hierbei nicht um C#, sondern um JavaScript handelt, zu dessen Möglichkeiten bereits eine ordentliche Dokumentation existiert [9]. Der sehr rudimentäre Editor im Browser speichert zwar auch anstandslos fehlerhaften Code, bietet allerdings durchaus Hinweise in Form unterringelter Zeilen und Tooltipps, wenn man ungültige Bezeichner verwendet. In Zukunft soll es für die Verwaltung der Mobile Services auch ein Management-API geben. Es ist zu hoffen, dass hierüber auch solche Skripte komfortabler gewartet werden können.

Beim serverseitigen Code greift ebenfalls der Mechanismus von Dynamic Schema und fügt die CreatedAt-Spalte automatisch hinzu, wenn sie noch nicht existiert. Das Feature erkennt, dass es sich um einen Datumswert handelt, und legt den Typ der Spalte korrekt fest. Bemerkenswert ist, dass unsere Clientanwendung weiterhin fehlerfrei funktioniert, obwohl die Definition der Thought-Klasse noch nicht angepasst wurde und dort die neue Spalte als Property noch gar nicht vorhanden ist. Derartige nicht definierte Daten werden einfach im Prozess der Deserialisierung verworfen. Um sich aber die neuen Informationen zunutze machen zu können, muss man abschließend natürlich den Clientcode ebenfalls anpassen.

## Authentifizierung und Autorisierung

Wenn man sich die serverseitigen Skripte etwas genauer ansieht, stellt man fest, dass neben dem eigentlichen Datenobjekt und der Operation in Form eines Requests auch ein User-Objekt in die Funktionen hereingereicht wird. Das dient dazu, dass neue Daten mit bestimmten Benutzern verknüpft werden können, es kann aber auch für zusätzliche Logik genutzt werden, um beispielsweise sicherzustellen, dass Benutzer nur ihre eigenen Daten bearbeiten und abrufen können. Die zur Authentifizierung unterstützten Dienste sind momentan Windows Live (Microsoft Account), Facebook, Twitter und Google. Da alle auf gleichen Konzepten beruhen (Stichwort OpenID/OAuth), wird hier stellvertretend nur die Integration mit Windows Live gezeigt. Hierzu muss zunächst einmal die Beispiel-Windows- Store-App im Live Connect Developer Center [10] bekannt gemacht werden. Wer seine App bereits im Windows Dev Center [11] registriert hat, sollte diese Verknüpfung bereits automatisch erledigt haben und im Live Connect Developer Center einen entsprechenden Eintrag vorfinden. Eine Anleitung zur manuellen Verknüpfung ist online verfügbar [12]. Danach beschränkt sich die Konfiguration darauf, in Live als Redirect-Domain die Mobile Services einzutragen (diese kann man dem Dashboard seines Mobile-Services-Projekts entnehmen, sollte man sie einmal vergessen haben) und im Gegenzug im Bereich Identity der Mobile Services das Client-Secret einzutragen, das man den API Settings aus Live entnimmt. Durch diese gegenseitige Bekanntmachung können die Mobile Services nun Windows Live zur Authentifizierung nutzen.

Zum Testen kann man beispielsweise die Zugriffsrechte für alle Operationen der Thoughts-Tabelle aus dem Beispiel auf Only Authenticated Users stellen. Beim nächsten Start der Anwendung wird das zu einem HTTP-Fehler 401 (Unauthorized) führen. Um dies zu beheben, benötigt man das eingangs erwähnte Live SDK, um den Client zu erweitern. Die Authentifizierung ist dann eine Sache weniger Zeilen Code, mit deren Hilfe man zunächst einen Login per Live ID veranlasst und ein Authentification Token anfordert, das man im zweiten Schritt zur Authentifizierung bei den Mobile Services verwendet (siehe Listing 1).

**Listing 1**

*Authentifizierung per Live ID.*

```
private static async Task<MobileServiceUser> Authenticate()
{
        // log on with Live ID
        var liveAuthClient = new LiveAuthClient("https://braindump.azure-mobile.net/");
        var liveLoginResult = await liveAuthClient.LoginAsync(new[] {"wl.signin"});
        if (liveLoginResult.Status != LiveConnectSessionStatus.Connected)
```

```
        {
            // unauthenticated user
            return null;
        }

        // use the session token to authenticate to Mobile Services
        var authToken = liveLoginResult.Session.AuthenticationToken;
        var mobileServicesLoginResult = await App.MobileService.LoginAsync(authToken);

// mobileServicesLoginResult.UserId is filled now
return mobileServicesLoginResult;
}
```

Beim nächsten Start der App wird Windows 8 automatisch nachfragen, ob es der Anwendung erlaubt werden soll, den Benutzer einzuloggen. Nur wenn der Benutzer dies absegnet, wird das programmatische Einloggen gelingen (siehe dazu Abbildung 8).

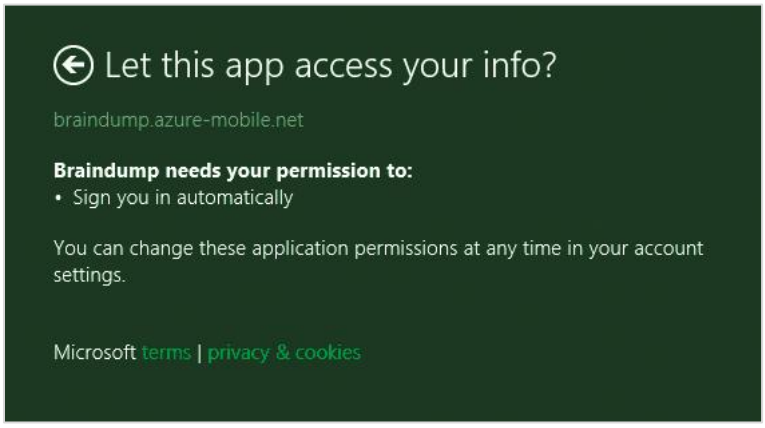



*Abbildung 8*

Einmal auf diese Weise bestätigte Apps kann der Benutzer in seinem Windows-Live-Profil online verwalten, um ihnen beispielsweise die Berechtigungen wieder zu entziehen.
Wurde die Erlaubnis erteilt, ist der Benutzer nicht nur im Client erfolgreich angemeldet. Die Daten stehen nun auch serverseitig zur Implementierung der zusätzlichen Logik in den Skripten zur Verfügung.

## Ausblick und Einschränkungen

Neben den oben beschriebenen bieten die Mobile Services auch heute schon zusätzliche Funktionen an, die für App-Entwickler sehr interessant sind, beispiels- weise die vereinfachte Unterstützung von

Push Notifications für die Windows- und Apple-Welt, bei denen Anwender über Datenänderungen und Ähnliches benachrichtigt werden, auch ohne dass die App selbst läuft oder aktiv eine manuelle Anfrage an den Server gestellt werden müsste. Künftig werden sicherlich weitere Funktionen hinzukommen, von denen einige bereits vage angedeutet wurden.

Wer sich nun abschließend fragt, was genau denn an den Mobile Services das „Mobile" sein soll, steht mit dieser Frage nicht allein da. Diverse Entwickler, Blogger und Autoren haben die Namensgebung schon heiß diskutiert, weil es tat- sächlich keine wirkliche Einschränkung auf mobile Geräte gibt. Alle hier vorgestellten und auch sonst verfügbaren Konzepte lassen sich selbstverständlich auch mit stationären Clients nutzen. Warum Microsoft den Begriff Mobile Services gewählt hat, ist momentan unklar. Eventuell werden zukünftig Funktionen hinzukommen, die nur von mobilen Geräten aus Sinn ergeben.

Sind Mobile Services also der Heilige Gral, wenn es um die Entwicklung von Cloud-basierten Backends geht? Bei allem Komfort und dem schon jetzt soliden Funktionsumfang sollte man weder die Intention noch die offensichtlichen Einschränkungen von Mobile Services aus den Augen verlieren. Es geht darum, möglichst schnell einfache Apps auf eine solide Basis zu stellen. Insbesondere haben Mobile Services die folgenden Nachteile:

- Fast alle Operationen sind an Tabellen beziehungsweise an der Datenbank festgemacht. Es gibt keine davon los- gelösten Service-Aufrufe, zum Beispiel um Berechnungen durchzuführen, oder lediglich Workarounds, um solche zu realisieren. Einzige Ausnahme sind serverseitige Scheduler-Skripte, die an klassische Cronjobs erinnern.
- Pro HTTP-Request ist nur eine Insert-, Update- oder Delete-Operation möglich.
- Serverseitiger Skriptcode kann nicht wiederverwendet und muss gegebenenfalls in allen Operationen dupliziert werden.
- Es werden immer für alle Tabellen Endpunkte erstellt. Weder Einschränkungen noch Transformationen sind möglich.

Es wird also deutlich, dass spätestens bei steigendem Komplexitätsgrad und erweiterten Anforderungen, die sich nicht in Microsofts vorgesehenes, momentan sehr vereinfachendes Schema pressen lassen, die manuelle Implementierung von Features eventuell die bessere Wahl ist. Für Hobbyentwickler oder Kleinprojekte sind die Mobile Services aber eine interessante Wahl, um in kürzester Zeit recht erstaunliche Ergebnisse auf die Beine zu stellen.

[1] Windows Azure Benefits für MSDN-Abonnenten, www.dotnetpro.de/SL1303AzureMobile1

[2] Visual Studio 2012 Downloads, www.dotnetpro.de/SL1303AzureMobile2

[3] Mobile Services SDK Download, www.dotnetpro.de/SL1303AzureMobile3, auch als Quelltext verfügbar unter
www.dotnetpro.de/SL1303AzureMobile4
[4] Download Live SDK, www.dotnetpro.de/SL1303AzureMobile5
[5] Partnerschaft Microsofts mit Xamarin für iOS- und Android-Support, www.dotnetpro.de/SL1303AzureMobile6
[6] Dokumentation des REST-API, www.dotnetpro.de/SL1303AzureMobile7
[7] Lizenzbestimmungen der Windows Azure Preview Features, www.dotnetpro.de/SL1303AzureMobile8
[8] Preisliste für die Mobile Services, www.dotnetpro.de/SL1303AzureMobile9
[9] Dokumentation der serverseitigen Scripting- Möglichkeiten, www.dotnetpro.de/SL1303AzureMobile10
[10] Live Connect Developer Center, https://manage.dev.live.com
[11] Windows Dev Center, https://appdev.microsoft.com/StorePortals/
[12] Anleitung zur manuellen Verknüpfung von Live und Mobile Services, www.dotnetpro.de/SL1303AzureMobile11

Mit freundlicher Unterstützung von:


www.dotnetpro.de

## Kapitel 6: Office Development

### Thorsten Hans (Experts Inside GmbH) – SharePoint-Apps mit Napa entwickeln: Entwickeln in der Wolke

#### Arbeitgeber

• Experts Inside GmbH

#### Technologieschwerpunkte

• SharePoint MVP
• SharePoint Development
• Web Development (CoffeeScript, ASP.NET MVC)
• Cloud based Solutions with Windows Azure

#### Mein Urlaubstipp

San Francisco ist für mich das Top Urlaubsziel, die Stadt verbindet fast 365 Tage schönes Wetter mit dem kalifornischen Spirit. Abseits der Touri-Routen kann man viele Überbleibsel der Flower-Power-Generation finden. Durch die direkte Meereslage kann man in San Francisco auch einen schönen Badeurlaub machen. Sehenswürdigkeiten bietet die Stadt in vollem Maße, daher ein guter Mix für einen perfekten Urlaub.

#### Meine Inseltechnologie

Im vergangenen Jahr hat sich im SharePoint Umfeld sehr viel getan, gerade die nahtlose Integration von Windows Azure bietet SharePoint Entwicklern mittlerweile die Möglichkeit die neusten Tools und Frameworks einzusetzen. Daher sind für mich die beiden Technologien SharePoint und Windows Azure absolut richtungsweisend in diesen Tagen.

# SharePoint-Apps mit Napa entwickeln: Entwickeln in der Wolke

Parallel zu Windows 8 und Windows Phone 8 hat Microsoft auch seine Unternehmensprodukte als App-Plattform ausgebaut. Das verändert das Erstellen von SharePoint-Erweiterungen.

Mit der jüngsten Version von Office, SharePoint und Office 365 können Softwareentwickler auf ein neues Programmiermodell zurückgreifen: auf Apps. Dies eröffnet jedem Entwickler viele neue Märkte, und dank Stores und Infrastrukturen seitens Microsoft kann jeder Entwickler ohne Mehraufwand eine viel größere Zahl an potenziellen Interessenten erreichen.

Natürlich lassen sich Apps für SharePoint 2013 (inklusive SharePoint Online) und Office 2013 weiterhin mit Visual Studio entwickeln. Dies ist auch in vielen Szenarien die einzige von Microsoft bereitgestellte Möglichkeit. Doch wenn Sie eine neue App schreiben, die nur aus clientseitigen Technologien besteht, also HTML, JavaScript und CSS, können Sie die neue Cloud-IDE Napa verwenden. Diese soll der Artikel grundlegend erläutern und anhand von zwei kleinen Beispielanwendungen veranschaulichen.

Um Napa zu verwenden, ist ein Office-365-Account nötig. Hierbei muss die Office-365-Instanz auf den 2013er Serverprodukten laufen; alle vorhandenen (alten) Office-365-Instanzen werden derzeit von Microsoft aktualisiert.
Wer keinen Office-365-Account hat, kann einen kostenlosen Probezugang beantragen [1]. Mit diesem lässt sich alles nachvollziehen, was in diesem Artikel gezeigt wird.

Weil es sich bei Napa um ein Entwicklungswerkzeug handelt, kann die App nicht auf jeder beliebigen SharePoint-Website-Sammlung installiert werden. Neu in SharePoint 2013 ist die Website-Vorlage Developer Site; eine Website- Sammlung dieser Vorlage ist die Voraussetzung, wenn Developer-Apps aus dem SharePoint Store installiert werden sollen.

## Anlegen einer Entwicklerwebsite in Office 365

Nach der erfolgreichen Anmeldung am Office- 365-Portal mit administrativen Rechten müssen Sie zunächst die SharePoint-Verwaltung über ADMIN | SHAREPOINT wie in Abbildung 1 aufrufen. Unter dem Punkt Site Collections lassen sich sämtliche Website-Sammlungen des Inhabers verwalten. Über die Schaltfläche NEW | PRIVATE SITE COLLECTION können Sie eine neue Website-Sammlung in SharePoint anlegen. Im dazugehörigen Dialog ist es wichtig, die Vorlage Developer Site auszuwählen. Im deutschsprachigen Administrationsportal lautet der Name dieser Website-Vorlage Entwicklerwebseite.

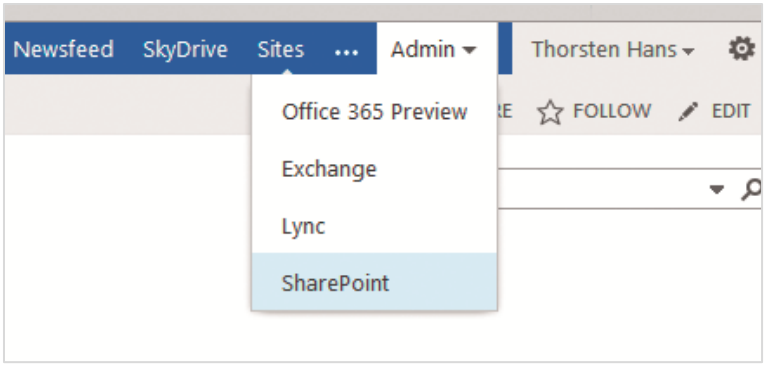



*Abbildung 1*

## Installieren von Napa

Ist die Website-Sammlung erstellt, muss diese nun aufgerufen werden. Über den Navigationspunkt SITE CONTENTS – zu finden im Quick-Launch-Bereich von SharePoint auf der linken Seite – können Sie vorhandene Apps aufrufen oder neue Apps zur Seite hinzufügen. Nach dem Klick auf Add an App ist auch der Zugriff auf den SharePoint Store möglich. Hier ist es wichtig, dass eine Region und eine Währung eingestellt sind, siehe Abbildung 2.

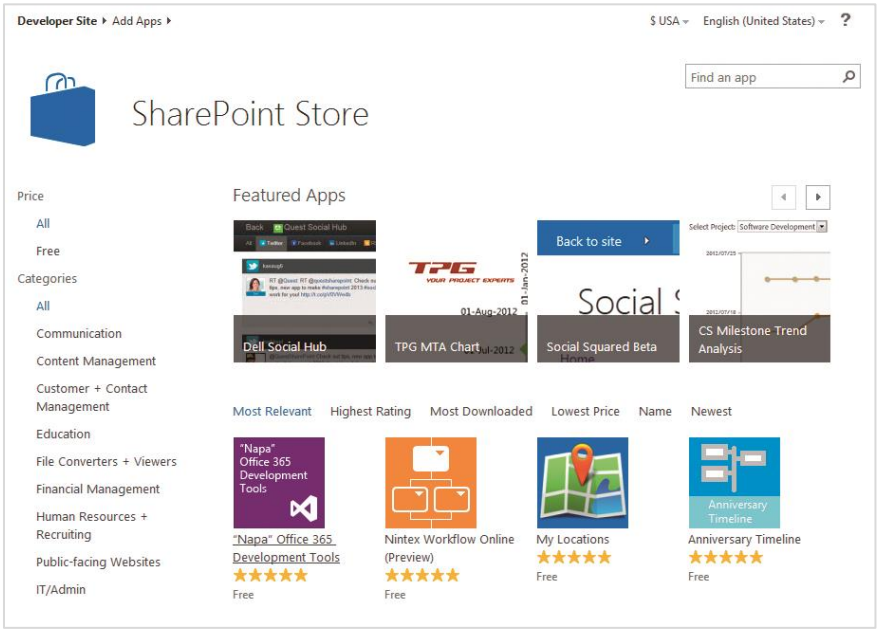



*Abbildung 2*

Dies gilt unabhängig davon, ob kostenlose oder kostenpflichtige Apps konsumiert werden. An dieser Stelle können Sie nun Napa auf der Entwicklerwebseite installieren. Dabei müssen der App entsprechende Rechte eingeräumt werden, damit sie ihre Aufgaben erfüllen kann (siehe Abbildung 3).


*Abbildung 3*

## Der erste Start

Wie jede App ist auch Napa über den Menüpunkt SITE CONTENTS aufzurufen. Da Sie noch keine Apps mit dieser Napa-Instanz entwickelt haben, wird Sie Napa nach dem ersten Start direkt nach dem Typ der zu erstellenden App fragen. Sobald mindestens eine App mit Napa erstellt wurde, präsentiert sich das Dashboard von Napa direkt nach dem Start der IDE. Aus dem Napa-Dashboard heraus lassen sich aktuelle App-Projekte öffnen oder neue starten. Darüber hinaus können Sie hier vorhandene App-Projekte auch kopieren, umbenennen oder löschen.

Napa bietet unterschiedliche App-Typen an. Abbildung 4 veranschaulicht, welche Typen unterstützt werden und wo diese im Zusammenspiel mit den Microsoft- Produkten angesiedelt sind.

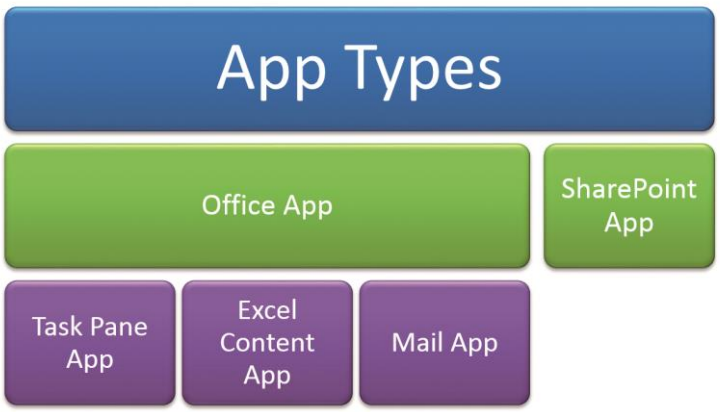



*Abbildung 4*

Für den SharePoint-Entwickler gibt es an dieser Stelle nicht viele Möglichkeiten; Napa erlaubt es hier, SharePoint-Apps zu erstellen. Anders sieht es aus, wenn Sie mit Napa Apps für Office oder Office-

Web-Apps entwickeln wollen. Hier unterscheidet Microsoft grundlegend zwischen drei Typen von Apps.

Task-Pane-App für Office: Sogenannte Task-Pane-Apps können in Napa derzeit für Word und Excel entwickelt werden. Die Task Pane selbst stellt hierbei die Optionen zur Oberflächengestaltung für die App bereit und wird am rechten Bildschirmrand der Office-Anwendung angedockt, wie es in Abbildung 5 zu sehen ist.

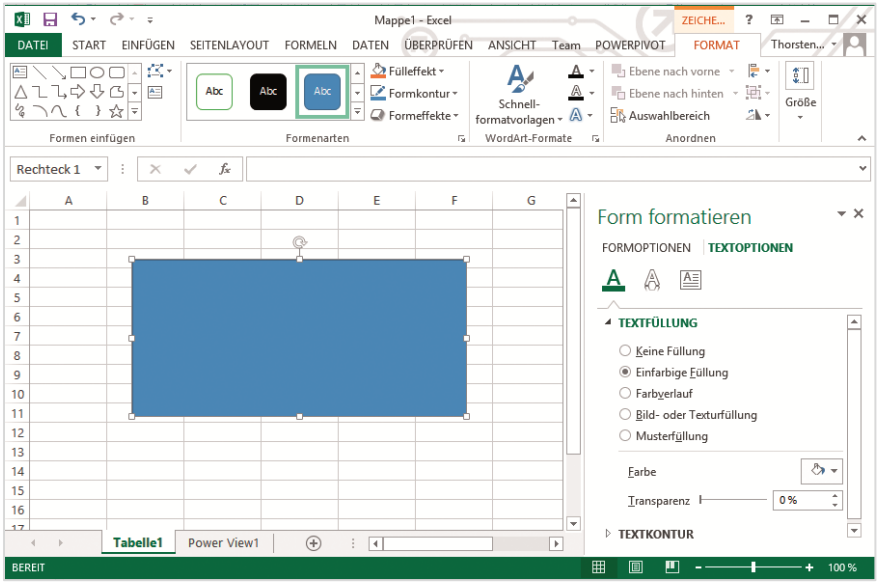



*Abbildung 5*

Content-App für Excel: Wie der Name schon sagt, können mit dieser Vorlage Apps für Excel entwickelt werden; hier- bei sind beide Excel-Varianten möglich, das normal auf einem PC installierte Excel und Excel als Webanwendung. Im Gegensatz zur Task-Pane-App wird diese Content-App direkt in das Tabellenblatt integriert.

Mail-App für Office: Zu guter Letzt steht auch eine Vorlage für Mail-Apps zur Verfügung, um Outlook zu erweitern. Hier- bei ist allerdings zu beachten, dass Napa derzeit lediglich die Erweiterung der Outlook-Web-App erlaubt; Apps für lokal auf einem PC laufendes Outlook sind zurzeit nicht vorgesehen.
Napa selbst kennzeichnet die unterschiedlichen App-Typen farbig, sodass schnell ersichtlich ist, um welchen Typ es sich handelt: für eine SharePoint-App das typische SharePoint-Blau, für eine Excel-Content-App das Excel-Grün, für eine Task-Pane-App das bekannte Office- Orange und für eine Mail-App das Blau von Outlook.com.

## Die Komponenten der Napa-IDE

Die Entwicklungsumgebung Napa selbst ist aufgeräumt, und so kann sich der Entwickler auf das Wesentliche konzentrieren – den Code. Abbildung 6 zeigt die wichtigsten Bestandteile der IDE.

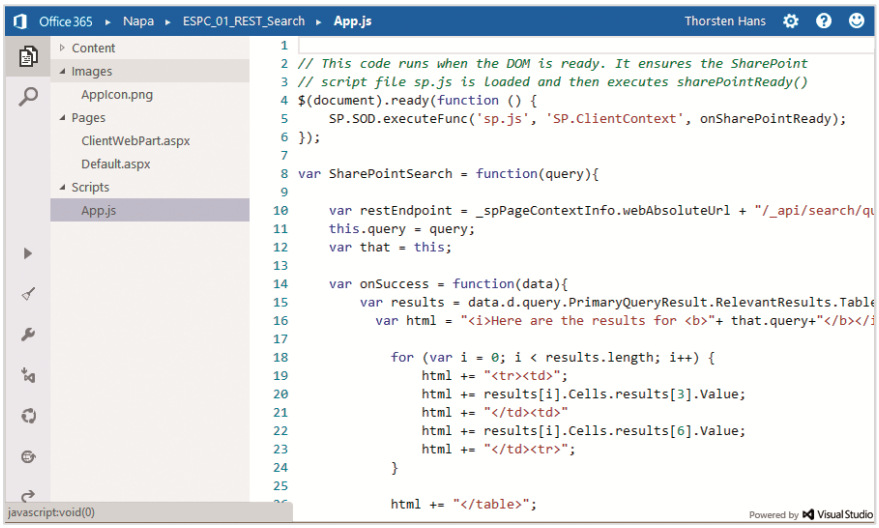



*Abbildung 6*

Der Code-Editor: Napas Code-Editor ist das Herzstück der IDE. IntelliSense, Syntaxhervorhebung, Schnellnavigation und grundlegende Codeassistenten sind vorhanden und machen das tägliche Entwickeln sehr flüssig. Gerade für neue JavaScript-Entwickler ist IntelliSense enorm hilfreich und erleichtert das Programmieren sehr. Auch die Unterstützung durch Assistenten ist vielversprechend. Schlechte Strukturen oder gar potenzielle Sicherheitslöcher bemängelt Napa unmittelbar. Abbildung 7 zeigt zum Beispiel, wie Napa auf die Verwendung der JavaScript-Methode eval() reagiert und wie IntelliSense hier aktiv wird.


*Abbildung 7*

Der App-Explorer: Der Bereich, der die Komponenten eines Napa-Projekts auflistet und der in Abbildung 7 links als Baumansicht zu sehen ist, ähnelt dem Projektmappen-Explorer von Visual Studio; der Bereich trägt keinen Titel, nennen wir ihn der Einfachheit halber App-Explorer. In diesem Explorer

stehen alle Funktionen zur Verfügung, die mit den Dateien der App in Verbindung stehen: neue Dateien hinzufügen, bestehende Dateien umbenennen, löschen, verschieben und so weiter. Microsoft bietet auch in Napa Item-Templates an.

Derzeit stehen drei Vorlagen zur Verfügung:

- JavaScript File
- Style Sheet
- SharePoint Webpage

Wichtig ist, beim Anlegen von neuen Dateien ein paar Regeln zu beachten. So müssen sämtliche Dateien mit einem Buchstaben beginnen. Außerdem unterstützt Napa nicht jede Dateierweiterung; die Dateierweiterung .json für JavaScript- Files etwa unterstützt die IDE erst seit wenigen Monaten.

Das Eigenschaftenfenster: Das Eigenschaftenfenster lässt sich in der Navigationsleiste über den Schraubenschlüssel aufrufen und ist neben dem Code-Editor wohl die wichtigste Komponente von Napa. Hier können Sie neben allgemeinen Angaben wie dem Namen der App und dem Pfad zum App-Symbol auch wichtige weitere Einstellungen vornehmen. Da je- de App um Berechtigung beim Anwender anfragen muss, ist hier exakt zu definieren, welche Berechtigungen die App zur korrekten Ausführung benötigt. Diese Angaben lassen sich im Eigenschaftenfenster wie in Abbildung 8 festlegen.

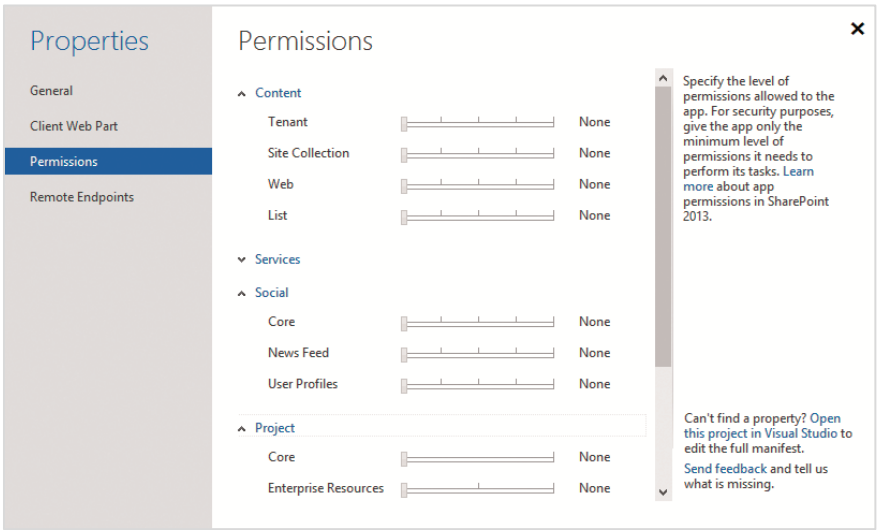



*Abbildung 8*

Ebenfalls bei den Eigenschaften angesiedelt sind die Punkte:

- Remote Endpoints
- Client Web Part
- Startup Page
- URL Parameters

## Die erste SharePoint-App

Nach dem Start der IDE müssen Sie zu- nächst im Dashboard eine neue Share- Point-App anlegen. Die folgende App soll unter Verwendung des SharePoint-REST- API die SharePoint-Suche nutzen und die gelieferten Ergebnisse in einfacher Form auf einer Webseite darstellen. Nach der Vergabe eines Namens für die App, hier ist es MySearchApp, wechseln Sie am besten ins Eigenschaftenfenster und legen dort die benötigten Berechtigungen fest.

Auf Abbildung 9 ist zu sehen, dass die Beispiel-App lediglich die Einstellung  Search – Query anfragt. MySearchApp besteht aus zwei Komponenten. Listing 1 zeigt die Präsentationsschicht in der Datei Default.aspx; hier werden lediglich die Elemente definiert. Das eigentliche Herz- stück der App aber wird in Form einer JavaScript-Klasse implementiert und ist in Listing 2 zu sehen. Unmittelbar vor der Klassendefinition ist noch der allgemeine JavaScript-Hook zu finden. SP.SOD sorgt dafür, dass die gewünschte JavaScript-Methode erst dann ausgeführt wird, wenn alle angegebenen Voraussetzungen verarbeitet wurden.

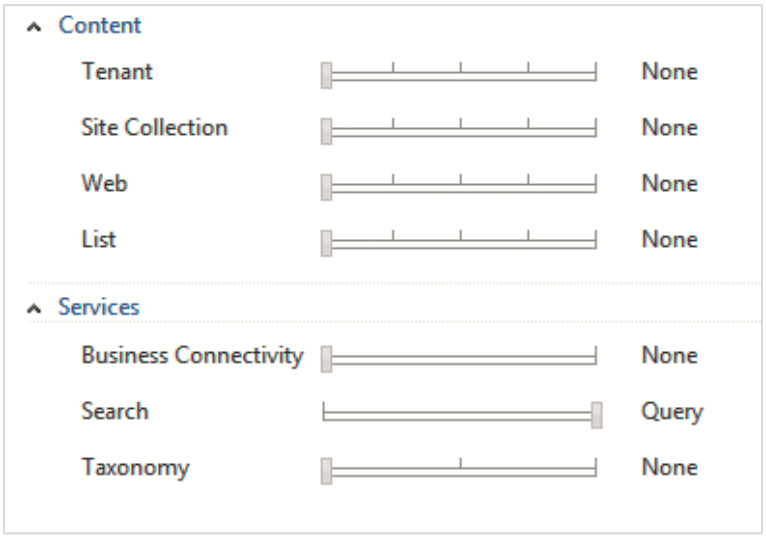



*Abbildung 9*

**Listing 1**

*Die Präsentationsschicht der Beispielanwendung MySearchApp.*

```
<!-- DEFAULT.aspx -->

<%-- The following 4 lines are ASP.NET directives needed when using SharePoint components --%>
<%@ Page Inherits="Microsoft.SharePoint.WebPartPages.WebPartPage, Microsoft.SharePoint,
        Version=15.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c"
    MasterPageFile="~masterurl/default.master" language="C#" %>
<%@ Register Tagprefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"
    Assembly="Microsoft.SharePoint, Version=15.0.0.0, Culture=neutral,
      PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="Utilities" Namespace="Microsoft.SharePoint.Utilities"
    Assembly="Microsoft.SharePoint, Version=15.0.0.0, Culture=neutral,
        PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="WebPartPages" Namespace="Microsoft.SharePoint.WebPartPages"
    Assembly="Microsoft.SharePoint, Version=15.0.0.0, Culture=neutral,
        PublicKeyToken=71e9bce111e9429c" %>

<%-- The markup and script in the following Content element will be placed in the <head> of the
    page --%>
<asp:Content ContentPlaceHolderId="PlaceHolderAdditionalPageHead" runat="server">
    <script src=https://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.6.2.min.js
        type="text/javascript"></script>
    <!-- Add your CSS styles to the following file -->
    <link rel="Stylesheet" type="text/css" href="../Content/App.css" />
    <!-- Add your JavaScript to the following file -->
    <script type="text/javascript" src="../Scripts/MySearchApp.js"></script>
</asp:Content>

<%-- The markup and script in the following Content element will be placed in the <body> of the
    page --%>
<asp:Content ContentPlaceHolderId="PlaceHolderMain" runat="server">
    <div>
        <input type="text" id="query"></input>
        <button type="button" id="doSearch">do search!</button>
        <hr/>
```

```
        <div id="results"></div>
    </div>
</asp:Content>
```

Listing 2 enthält einige interessante Aspekte, die sich wohl in jeder SharePoint- App wiederfinden. Dazu gehört beispielsweise die Verwendung der globalen Variable _spPageContextInfo. Diese eher unscheinbare Variable bietet wichtige Informationen wie zum Beispiel

- die aktuelle Sprache,
- den Site-URL,
- den Web-URL,
- die UI-Version,
- die User-ID.

Insbesondere die URLs sind wichtig beim Erstellen der Adressen für Ajax-Requests. Listing 2 enthält am Ende einen solchen, hierbei ist sehr wichtig, die Header-Angabe accept auf "application/ json;odata=verbose" zu setzen; andernfalls wird der Ajax-Aufruf kein Ergebnis liefern.

**Listing 2**

*Das JavaScript-Herzstück der App.*

```
//MySearchApp.js

$(document).ready(function () {
    // Die gewünschte Methode (onSharePointReady) wird erst dann ausgeführt, wenn
    // die SharePoint-eigene JS Lib (sp.js) geladen wurde
    // (SharePoint ScriptsOnDemand)
    SP.SOD.executeFunc('sp.js', 'SP.ClientContext', onSharePointReady);
});

function onSharePointReady(){
    $("#doSearch").click(function(event){
    var search = new SharePointSearch($("#query").val());
    search.doSearch();
    // Standard-Klick-Behandlung deaktivieren
    event.preventDefault();
    });
}
```

```
var SharePointSearch = function(query){

        // Die Variable _spPageContextInfo enthält sehr viele gute
        // Informationen, die im täglichen Alltagsgeschäft immer wieder
        // benötigt werden
        var restEndpoint = _spPageContextInfo.webAbsoluteUrl
            + "/_api/search/query?querytext='" + query + "'";

        this.query = query;
        var that = this;

        // Success-Handler
        var onSuccess = function(data){

            var results =
                    data.d.query.PrimaryQueryResult.RelevantResults.Table.Rows.results;
            var html = "<i>Here are the results for
                            <b>"+ that.query+"</b></i><table>";

            for (var i = 0; i < results.length; i++) {
                    html += "<tr><td>";
                    html += results[i].Cells.results[3].Value;
                    html += "</td><td>"
                    html += results[i].Cells.results[6].Value;
                    html += "</td><tr>";
                    }
            html += "</table>";
            $("#results").html(html);
        };
// Error-Handler
        var onError = function(error){
            $("#results").html("<b>Sorry, something went wrong ,"  + error + "'</b>");
        };

        // Öffentliche Methode zum Starten der Suche
        this.doSearch = function(){
```

```
        $.ajax({url: restEndpoint,
                method: "GET",
                headers: { "accept": "application/json;odata=verbose", },
                success: onSuccess,
                error: onError});
    };
};
```

Mehr ist nicht notwendig, um die erste SharePoint-App mit Napa zu erstellen. Abschließend genügt ein Klick auf die Play-Schaltfläche in der Navigationsleiste, damit Napa die App auf die dazugehörige Developer Site überspielen kann. Wie bei der Installation einer App aus dem SharePoint Store heraus sind auch hier der App die gewünschten Berechtigungen einzuräumen, sonst kann sie nicht auf der gewünschten Seite installiert werden. In Abbildung 10 ist MySearchApp in einer Office-365-Umgebung zu sehen.

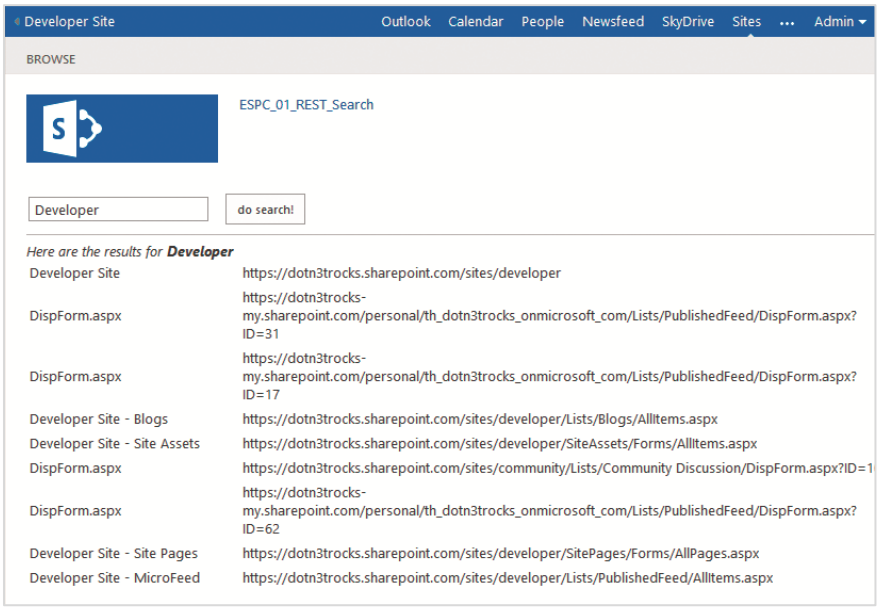



*Abbildung 10*

## Die erste Task-Pane-App

Eine einfache Task-Pane-App besteht wie eine SharePoint-App aus nur wenigen Komponenten. Neu für Office-Entwickler ist an dieser Stelle, dass sich nun auch die lokal installierten herkömmlichen Office-Anwendungen über HTML und JavaScript erweitern lassen; schon Gelerntes können Sie also getrost weiterverwenden.

Wechseln Sie zunächst ins Napa-Dashboard und erstellen Sie hier eine neue App vom Typ Task Pane App. Im Explorer finden Sie im Ordner HOME die beiden HTML- und JavaScript-Dateien, die in diesem

Beispiel angepasst werden. Die folgende Beispielanwendung stellt eine einfache Oberfläche in der Task-Pane-Leiste dar und konsumiert einen Web- dienst. Das Resultat des Webdienst-Aufrufs fügt die App dann an der Stelle der aktuellen Auswahl innerhalb des Dokuments oder der Tabelle ein.

Listing 3 zeigt den Inhalt der Datei home.html; die Logik wird wieder in die JavaScript-Datei in Listing 4 gepackt. Wie im Code zu sehen ist, wird die Methode setSelectedDataAsync() des Objekts Office. context.document aufgerufen; der Methodenname zeigt an, dass auch im Client konsequent asynchrone Methoden verfügbar sind, um dem Nutzer ein möglichst flüssiges Arbeiten zu ermöglichen.

**Listing 3**

*Die Datei home.html.*

```
<!-- HOME.html -->
<!DOCTYPE html>
 <html>
<head>
        <meta charset="UTF-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
        <title>RoundhouseKicker</title>
        <script src=https://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.1.min.js
            type="text/javascript"></script>

        <link href="../../Content/Office.css" rel="stylesheet" type="text/css" />
        <script src=https://appsforoffice.microsoft.com/lib/1.0/hosted/office.js
            type="text/javascript"></script>

        <link href="../App.css" rel="stylesheet" type="text/css" />
        <script src="../App.js" type="text/javascript"></script>

        <link href="Home.css" rel="stylesheet" type="text/css" />
        <script src="Home.js" type="text/javascript"></script>
</head>
<body>
        <!-- Page content -->
        <div id="content-header">
            <div class="padding">
```

```
            <h1>Roundhouse Kicker</h1>
        </div>
    </div>
    <div id="content-main">
        <div class="padding">
            <p><strong>Chuck Norris can split an atom. With his bare hands".</strong></p>
            <p>Or....<br/>insert another fact! :-)</p>
            <button id="InsertFactAtSelection">Insert Fact</button>
        </div>
    </div>
</body>
</html>
```

**Listing 4**

*Die JavaScript-Logik für home.html.*

```
// HOME.js
/// <reference path="../App.js" />
/*global app*/

(function () {
    'use strict';

    // The initializing function must be run each time a new page is loaded
    Office.initialize = function (reason) {
        $(document).ready(function () {
            app.initialize();
            $('#InsertFactAtSelection').click(insertAFact);
        });
    };

    function insertAFact() {
        $.getJSON("http://api.icndb.com/jokes/random?callback=?", function(res){
            Office.context.document.setSelectedDataAsync(res.value.joke, function(asyncResult){
                app.showNotification('Error:', asyncResult.error.message);
            });
        });
```

```
        }
})();
```

Schließlich können Sie im Eigenschaftenfenster noch angeben, in welcher Anwendung die App gestartet werden soll. Unter RUN können Sie zwischen den Zielanwendungen auswählen.

Nach dem Start der App präsentiert sich die erste Office-Task-Pane-App, der „Roundhouse Kicker", in Excel 2013, wie in Abbildung 11 zu sehen ist.

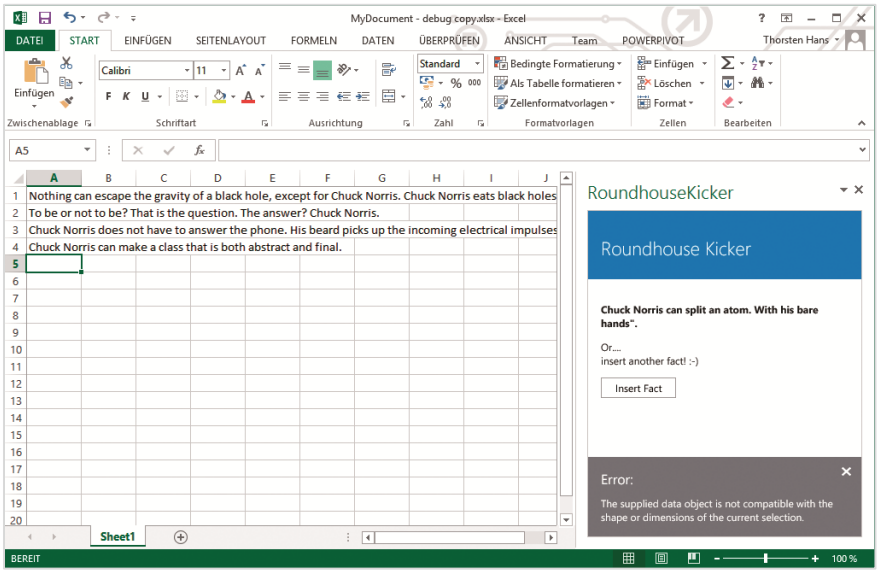



*Abbildung 11*

## Weitere Funktionen von Napa

Napa bietet noch einige weitere interessante Funktionen. Über die Share- Schaltfläche in der Navigationsleiste ist es möglich, den Code der App anderen Teilnehmern zugänglich zu machen. Der Betrachter einer freigegebenen App kann diese lediglich lesen; Anpassungen sind auf diese Weise nicht möglich. Über die gleiche Navigationsschaltfläche können Sie die Freigabe jederzeit auch wieder entziehen oder bei Bedarf eine neue Version des Quellcodes freigeben.

Microsoft hat der Mini-IDE auch eine dateiübergreifende Suche mitgegeben. Durch einen Klick auf das Lupensymbol in der Navigationsleiste oder mit der Tastenkombination [Ctrl]+[Shift]+[F] lassen sich alle Dateien des App-Projekts durchsuchen. Auf Wildcards und reguläre Ausdrücke müssen Sie dabei nicht verzichten. Für den Fall, dass Napa für das Projekt doch nicht genügen sollte, hat Microsoft die Schaltfläche Edit in Visual Studio eingebaut. Sie packt das aktuelle App-Projekt und bietet es zusammen mit dem Webplattform-Installer zum Download an. Der Installer kümmert sich darum, dass ein eventuell nagelneuer Rechner als Entwicklungsmaschine für SharePoint-Apps konfiguriert wird.

Somit sind auch für die Entwickler von SharePoint-Apps die Tage vorbei, in denen sie die Voraussetzungen dafür manuell suchen und einrichten mussten. Wenn das System dann vorbereitet ist, kann die Entwicklung der App naht- los auf dem lokalen Rechner weitergehen.

## Fazit

Das Nutzungsverhalten des Autors soll das Fazit zu Napa unterstreichen. In dessen Napa-Instanz liegen circa 60 App-Projekte für SharePoint, ergänzt durch einige Office-Apps. Das ist schon eine beachtliche Zahl und zeigt, dass Napa hier mittlerweile einen festen Platz unter den Entwicklungswerkzeugen eingenommen hat. Gerade für schnelles Prototyping, die Integration von Daten aus beliebigen Diensten, Feeds oder Webseiten ist Napa kaum zu schlagen. Natürlich ist der Integrations- und Interaktionsgrad mit dem SharePoint-API in diesen Beispielen nicht hoch; doch mithilfe der jüngsten Version des Objektmodells für den Client ist dies im Handumdrehen in Napa zu erledigen. Die Entwicklung mit Napa ist schnell, unkompliziert und agil. Wer sich darauf einlassen kann, auf die Client-Technologien beschränkt zu sein, für den wird Napa zunehmend wichtiger für das Erstellen von SharePoint-Apps.

[1] Apps für Office und SharePoint, http://dev.office.com

Mit freundlicher Unterstützung von:

dotnetpro

www.dotnetpro.de

## Senaj Lelic (oneAssist UG) – BI Lösungen mit Visio. Graphische Visualisierungen mit Visio, SharePoint und Visio Services

### Arbeitgeber

• oneAssist UG

### Technologieschwerpunkte

• Visio, SharePoint, Office und SharePoint Apps

• Microsoft Technologien

### Mein Urlaubstipp

Kroatien, die istrianische Küste, grün auch im Sommer und dennoch warm.

### Meine Inseltechnologie

BI Lösungen mit Visio. Graphische Visualisierungen mit Visio, SharePoint und Visio Services.

## BI Lösungen mit Visio. Graphische Visualisierungen mit Visio, SharePoint und Visio Services

Mehr oder weniger jeder Office Anwender im Unternehmen kennt Visio oder lernt es irgendwann kennen. Dabei fällt immer früher oder später der Begriff „Maltool" oder „Malwerkzeug". Manch einer fängt dann an und „malt" mit Visio dann seine Netzwerkdiagramme, Organigramme etc.

Dabei könnte man mit dem Begriff „Malwerkzeug" Visio kam mehr unrecht tun – denn wenn Visio genau eines NICHT ist, dann das: ein Malwerkzeug. Tatsächlich kann jedes gemeine einfache Malprogramm mehr als Visio in Bezug auf Malen und Zeichnen. Nein, das was man mit Visio erstellt ist

genau genommen überhaupt keine Zeichnung, denn diese zielt ja darauf ab, einen Aspekt der Realität möglichst gut bzw. detailliert abzubilden. Wer sich jedoch mal die Darstellungen von Visio ansieht, dem fällt schnell auf, dass die Darstellungen in Visio mit Realität in Bezug auf die Darstellung, das Aussehen, so gar nichts zu tun haben.

In jeder Organisation gibt es Darstellungen bei denen es genau genommen überhaupt nicht um das Aussehen geht, sondern viel mehr um Daten und Datenzusammenhänge. Der Unterschied zur Zeichnung ist darüber hinaus auch, dass diese Darstellungen höchst standardisiert sind und man schon vom „Hören" her weiß, worum es geht. Wenn der Begriff Organigramm fällt, dann überlegt sicher auch niemand, was denn das nun wäre. Der Grund ist, dass diese Darstellungen höchst vereinheitlicht sind und überall gleich aufgebaut werden. Ein Organigramm beispielsweise läuft immer von oben nach unten, besteht aus Kästchen und zeigt die hierarchische Struktur einer Organisation – davon weicht kaum jemand ab.

Diese Darstellungen (zu denen auch Netzwerkabbildungen, oder Flussdarstellungen gehören) bezeichnet man als Diagramm – bei dem es eben um bestimmte Aspekte oder Datenzustände geht – nicht jedoch um die Präzision der Darstellung – und genau das ist Visio: ein Diagrammtool. Somit vermisst man bei Visio vieles, was jedes „Zeichenprogramm" hat, weil es eben für Diagramme gedacht ist.

Denkt man nun an BI(Business Intelligence) -Szenarien, also eine Darstellung die mir in wenigen Augenblicken hilft, geschäftskritische Entscheidungen zu fällen, so liegt es nahe hier Visio einzusetzen. Dies insbesondere, weil diese Darstellungen ebenso höchst standardisiert sind, und es auf die Hervorhebung oder Visualisierung von bestimmten Daten bzw. Datenzuständen geht, weniger um komplexe Abbildungen. Visio ist also das einfachste und dennoch mächtige BI-Tool aus dem Hause Microsoft – eigentlich aber ein allgemeines Diagrammtool, das eben „noch nebenbei" für BI-Zwecke genutzt werden kann.

## Diagramme: die Power von Visio

Geschäftsdiagramme sind also Darstellungen bei denen es darum geht in wenigen Augenblicken einen Sachverhallt dazustellen, Daten und Datenzusammenhänge abzubilden. Damit Visio diese Aufgabe erfüllen kann, fehlt noch eine Komponente: die Daten. Visio-Diagramme unterscheiden sich von Zeichnungen auch dadurch, dass Visio-Dateien/Diagramme die Fähigkeit haben, die Daten direkt in das Diagramm mit hineinzuladen anstatt diese „hineinmalen" zu müssen. Die Komponenten von Visio aus denen Diagramme entstehen, die Shapes (alles auf einem Visio-Diagramm-Blatt heißt „Shape")

haben auch Container in denen man eine nahezu beliebige Menge von Daten aufnehmen kann. Visio-Diagramme sind also immer auch datengestützte Diagramme. Shapes sind gewissermaßen „optische Container" für Daten und Datenpakete. Bei einem Netzwerk-Server-Shape wäre diese Datenmenge beispielsweise bestehend aus: Inventarnummer, IP-Adresse, Hauptspeicher, Festplattenkapazität etc., etc. Je nach Diagrammart und Anwendungsgebiet variiert diese Menge – und deswegen ist es auch sehr einfach in Visio die Datenstruktur nach Bedarf anzupassen.

## Datengestützte Diagramme: die Grundlage jeder BI

Die Daten und deren Struktur sind also diagrammspezifisch und anpassbar. Jedoch hilft es mir wenig wenn der Aufbau und die Pflege des Diagramms sehr viel Zeit in Anspruch nimmt – soll doch das Diagramm nicht der Hauptzweck meiner Arbeit sein, sondern eine unterstützende Komponente, damit ich meine Arbeit schneller erledigen kann. Besonders hinderlich und fehleranfällig ist die Datenpflege im Diagramm, also das eintragen und ändern. Damit dies für den Anwender einfach bleibt, bietet Visio seit der Version 2007, Edition Professional den Datenverbindungsassistenten. Eine Software-Komponente, die aus nahezu jeder beliebigen strukturierten Datenquelle die Daten in Visio-Shapes übertragen kann. Dabei ist der Prozess ein rein lesender Prozess im Bezug auf die Datenquelle, es besteht also keine Gefahr die Datenquelle zu kompromittieren.

Visio unterstützt folgende Datenquellen „out of the box":

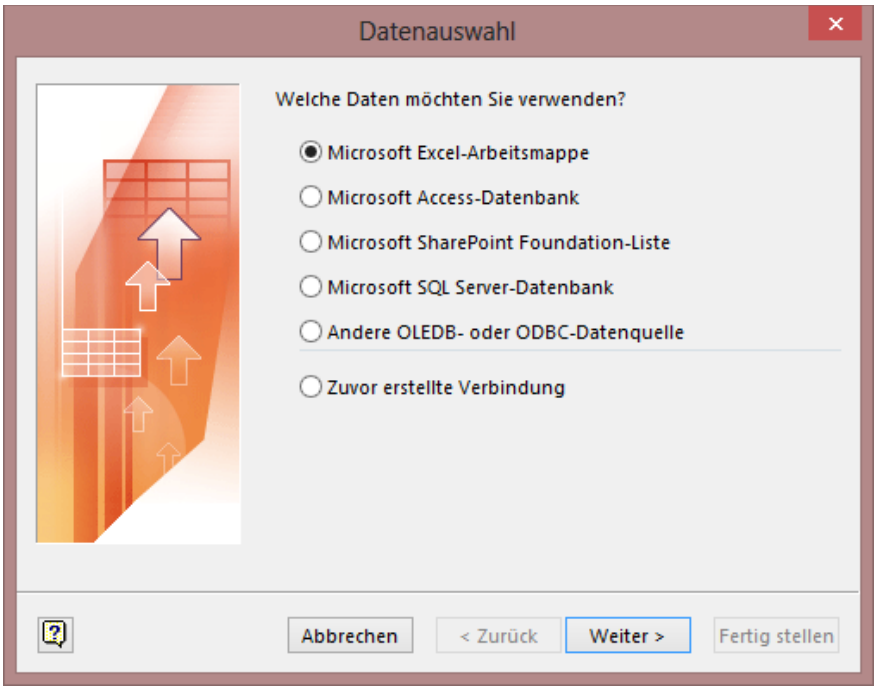



*Abbildung: Datenquellen*

Man sieht deutlich, dass hier so ziemlich alles, was sich „strukturierte Datenquelle“ nennt, enthalten ist, stehen doch über ODBC und OLE DB auch alle nicht direkt unterstützten Datenquellen zur Verfügung.
Ist das Shape (per Drag & Drop) dann mit einem Datensatz verknüpft, so ist die erste große Hürde genommen: das Diagramm ist da und die Daten sind in der Container-Komponente enthalten.
Dies hilft jedoch für BI-Aufgaben noch recht wenig, da ja oftmals nicht die originären bzw. Rohdaten interessant sind. Vielmehr möchte man Trends oder Gruppierungen oder „Ausreißer“ erkennen und auf einen Blick identifizieren, um bei Bedarf sofort Maßnahmen ergreifen zu können.

## Datenvisualisierung und -interpretation: die Datengrafiken

Ebenso mit Visio 2007 Professional Edition (und neuer aber immer Professional Edition) bringt Visio die Datengrafiken mit. Das sind Visualisierungs- und Interpretationsregeln die an beliebige Shapes angewendet werden können und einmal zentral definiert werden.
Jede Datengrafik benötigt zwei Dinge: das Datenfeld welches ausgewertet werden soll und die Interpretationsart.

Folgende Interpretationskategorien stehen zur Verfügung:

- Text: hier wird nicht interpretiert sondern der Wert als Wert angezeigt (Label)
- Datenbalken: bei Zahlenwerten kann in einem Datenbalken (Thermometer z.B) der Wert auf einer Skala zwischen Minimum und Maximum dargestellt werden
- Satz von Icons: hier wird einem Wert oder Wertebereich ein Icon zugewiesen
- Farbe nach Wert: für einen konkreten Wert wird eine bestimmte Füllfarbe zugewiesen

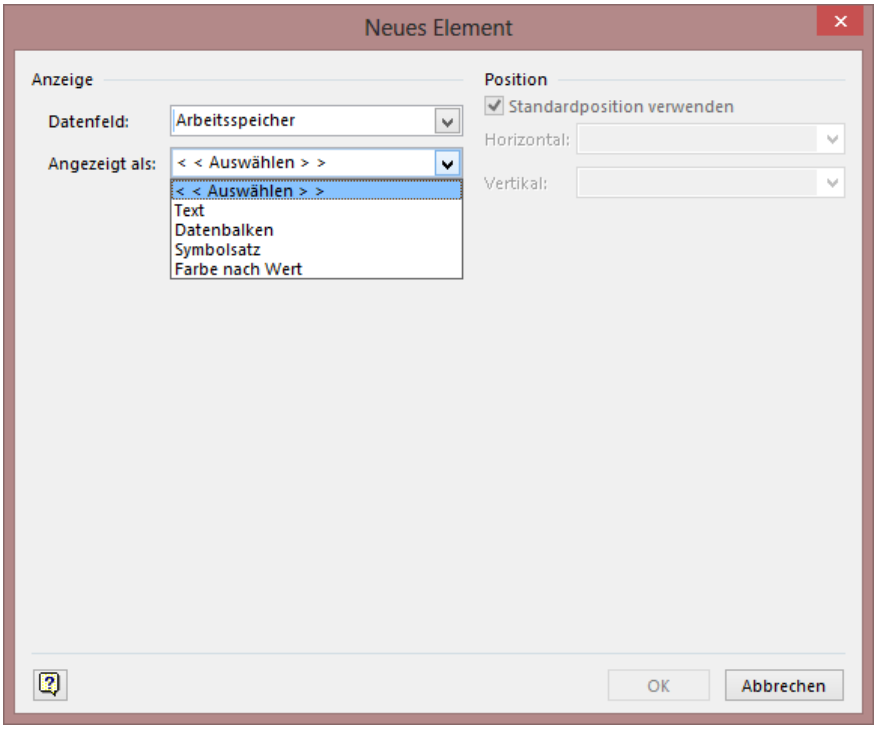


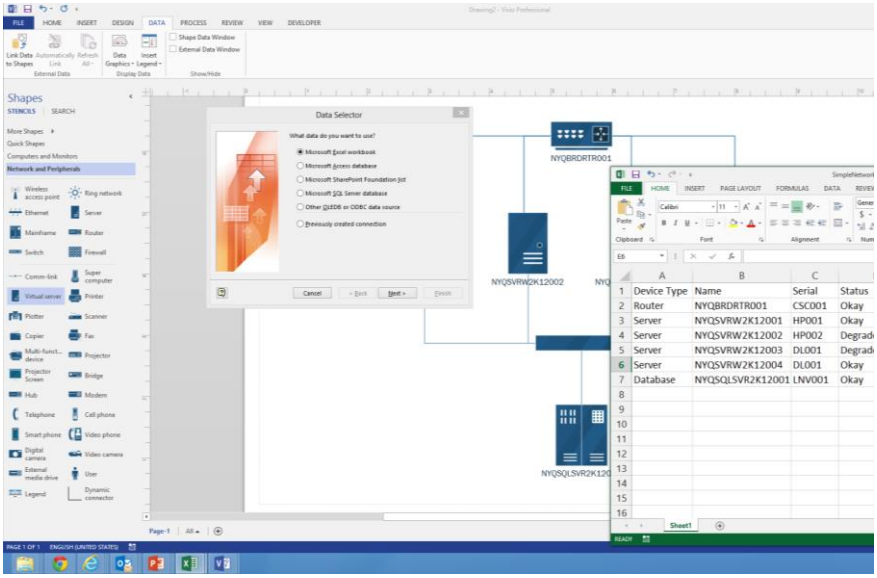

*Abbildung: Datengrafiken in Visio – die Kategorien*

Wendet man eine Datengrafik auf ein Shape an – dabei kann ein Shape bei mehreren Datenfeldern durchaus mehrere Datengrafiken haben – so erscheint die Datengrafik um das Shape herum bzw. färbt bei der letzteren das gesamte Shape entsprechend.

## Vom Diagramm zur BI-Grundlage

Das so erstellte, datengebundene und mit Datengrafiken erweiterte Diagramm ist somit schon die BI-Grundlage erlaubt es doch, auf einen Blick die kritischen Werte (als Icon z.B.) zu erkennen bzw. bei Häufung von Icons auch Trends zu erkennen.

„Ein Blick sagt mehr als 1000 Worte" ist zum einen der Slogan von Visio und zum anderen auch eine sehr treffende Bezeichnung für BI-Anforderungen.



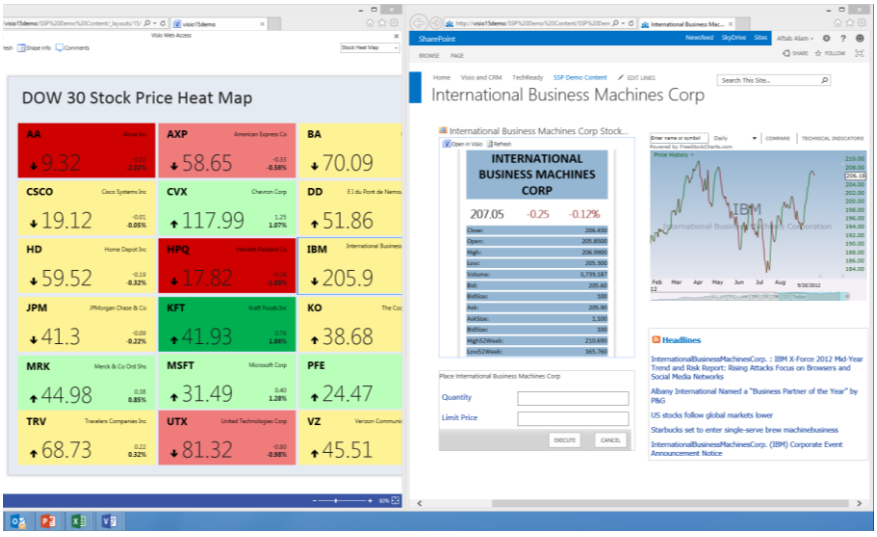

*Vom Diagramm zur Entscheidungsgrundlage: Publikation in Visio Services*

Nun hat man ein datengestütztes Diagramm und möchte dieses zur Arbeits- und Entscheidungs-grundlage für einen größeren Kreis von Anwendern machen. Die auf den ersten Blick einfachste Option wäre, den Anwendern die Datei zu geben. Das ist jedoch oftmals wenig sinnvoll. Zum einen haben die Anwender (auch teils gewollt) keinen direkten Zugriff auf die Datenquelle, zum anderen möchte man die Datei auch nicht im Original herausgeben, da somit Manipulationen möglich wären und zum Dritten hat nicht jeder Anwender zwangsläufig Visio. Ein Export der Visio-Datei in Standard-Graphikformate oder HTML ist oftmals wenig sinnvoll da diese statisch sind und somit immer nur eine Momentaufnahme zeigen.
Mit SharePoint bringt Microsoft seit Version 2010 und Edition Enterprise eine Komponente mit, die diese Probleme alle löst: Visio Services.

Visio Services ist ein Bestandteil von SharePoint Enterprise und ermöglich die Publikation von Visio-Dateien nach SharePoint. Dabei wird technisch die Visio-Datei einfach in eine SharePoint-Dokumentbibliothek gespeichert. Die Anzeige für den „normalen“ Anwender erfolgt dann in einem speziellen SharePoint-Webpart, dem Visio-Web-Access Webpart. Dieses kann die Visio-Datei im Browser ohne Zusatzkomponenten anzeigen und dem Anwender zum „gucken aber nicht anfassen“ bereitstellen. Dabei ist die Anzeige kein Export, der Anwender sieht die Originaldatei, kann sie aber nicht verändern.

Wichtiger Unterschied zu einem normalen Viewer ist, dass die Visio Services die Datenverbindung zur Datenquelle in der Visio-Datei auswerten können und die Daten im Diagramm aktualisieren können – on the fly. Die Diagramme im Portal sind also immer datenaktuell, und die Datengrafiken werden auch live aktualisiert.

Solche Diagramme und Visualisierungen lassen sich also erzeugen ohne eine einzige Zeile Code geschrieben zu haben – was gerade in Zeiten sich dauernd ändernder Anforderungen ein großer Pluspunkt ist. Der Anwender der die Darstellungen erzeugt muss ebenso wenig Programmierer sein wie der Endanwender – der erstere muss nur mit Visio umgehen können.




*Beispielhafte Visualisierung im Visio Web Access Webpart*

# Kapitel 7: Windows 8 Development

## Matthias Jauernig (SDX AG) – Windows 8 App Entwicklung

### Arbeitgeber

• SDX AG

### Technologieschwerpunkte

• Windows 8

• WinRT

• C#/XAML

• JavaScript/HTML5

• ASP.NET MVC

### Mein Urlaubstipp

Schweden. Das Land der Weite, Wälder und Seen lädt im Sommer zum Baden, Entspannen und Natur-Genießen ein.

### Meine Inseltechnologie

Windows 8-Apps werden immer populärer und laufen auf unterschiedlichsten Geräten. Umso wichtiger ist es, als Entwickler am Ball zu bleiben und zu verstehen, was die neue App-Welt ausmacht und wie sich ausgezeichnete Apps komfortabel entwickeln lassen.

## Mit Windows Apps in die Zukunft

Mit Windows 8 wird eine neue Klasse von Anwendungen Einzug in das Windows-Universum halten: Windows Apps. Sie lassen sich besonders gut per Touchscreen bedienen.

Auf der BUILD-Konferenz [1] hat Microsoft im September 2011 einen ersten detaillierten Blick auf das kommende Windows 8 gegeben. Es soll sowohl auf Tablets als auch auf den altbekannten Desktop-PCs lauffähig sein. Dazu steht der klassische Windows-Desktop zur Verfügung, der per Tastatur und Maus bedient werden kann und nicht grundlegend verändert wurde. Anders sieht das bei dem neuen UI aus, dessen Kacheloptik bereits von Windows Phone beziehungs- weise der Xbox bekannt ist. Für dieses UI kann mit den Windows-Apps eine neue Art von Windows-Anwendungen entwickelt werden. Sie können zwar immer noch per Tastatur und Maus bedient werden, eignen sich allerdings vor allem für die Touch-Eingabe.

Während klassische Desktop-Anwendungen wie bisher unter anderem mit dem Win32-API, .NET oder Silverlight entwickelt werden können, steht zur Entwicklung von Windows-Apps ein neues Programmiermodell zur Verfügung, das dotnetpro bereits in [2] vorgestellt hat. Der vorliegende Artikel baut auf diesen Grundlagen auf und beschreibt anhand einer Beispielanwendung, was in der Praxis auf Entwickler von Windows-Apps zukommt.
Der Beitrag basiert auf der Windows 8 Developer Preview [3], die Microsoft zur BUILD zum freien Download bereitgestellt hat, einige Angaben können sich daher noch ändern. Auf vorhandene Fehler der Developer Preview geht dieser Artikel explizit nicht ein, da zu vermuten ist, dass diese noch behoben werden. Die aktuelle Roadmap von Windows 8 sieht vor, dass die Beta Ende Februar erscheint, woraufhin mit der finalen Version im Sommer oder Herbst 2012 zu rechnen sein dürfte.

## WinRT – das Fundament

Das Fundament für Windows Apps ist die Windows Runtime (WinRT), die mit Windows 8 eingeführt wird [2]. Zur Wiederholung sei erwähnt, dass es sich bei WinRT um eine moderne Ablösung des Win32-API handelt, das in nativem C++ entwickelt wurde und Betriebssystem- Funktionalität für die unterstützten Sprachen bereitstellt. WinRT bildet den Kern des neuen Programmiermodells, siehe Abbildung 1 [4].

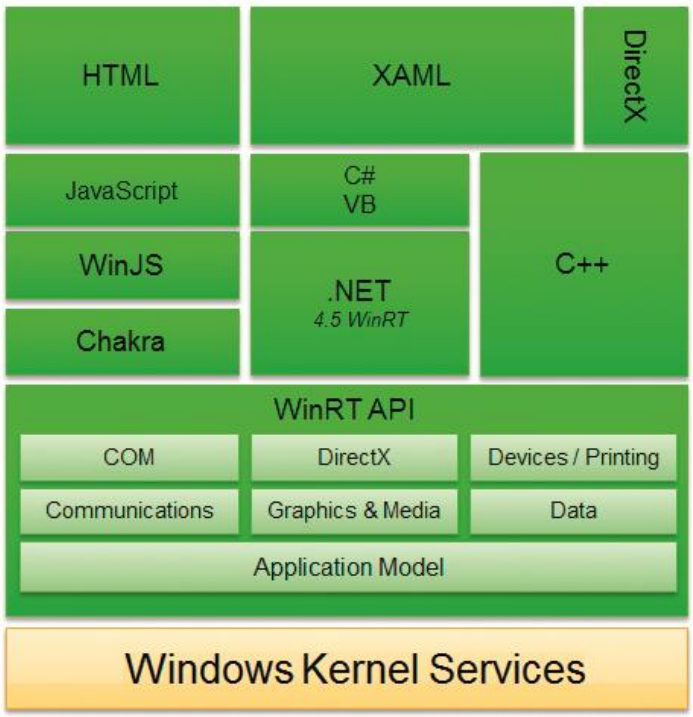



*Abbildung 1*

Aufbauend auf WinRT können Windows Apps in den unterstützten Sprachen XAML/C#, XAML/ VB, HTML5/JavaScript und XAML/C++ respektive DirectX/C++ entwickelt werden. Die native Implementierung ermöglicht performante Windows Apps, die auch auf schwächeren Rechnern flüssig ausgeführt werden.

Als Szenario zur Entwicklung einer ersten Windows App habe ich einen RSS-Reader implementiert, der die Beiträge des Corporate Blogs der SDX AG [5] aufbereitet darstellt. Das primäre Ziel war das Sammeln erster Erfahrungen in der Entwicklung von Windows Apps unter Windows 8. Für die Implementierung habe ich die Developer Preview von Visual Studio 2012 auf Basis der WinRT verwendet. Zum Einsatz kamen sowohl XAML/C# als auch HTML5/JavaScript.

Visual Studio 2012 unterstützt die Entwicklung von Windows Apps mit Projekt-Templates, die bereits einen möglichen Rahmen für Inhalte einer App bereitstellen. Diese Templates stehen für alle unterstützten Sprachen zur Verfügung.
Für die SDX-News-App fiel die Wahl auf das Grid Application-Template. Es stellt auf einer Master-Seite eine Anzahl von Elementen überblicksartig als gruppiertes Grid dar. Mit Touch-Gesten oder der Maus lässt sich horizontal durch das Grid scrollen. Bei der SDX-News-App handelt es sich um die einzelnen Blogbeiträge, die jeweils mit einem Bild und gruppiert nach dem Monat der Veröffentlichung angezeigt werden, siehe Abbildung 2. Tippt oder klickt der Nutzer auf ein Element, so bietet das

Template eine Vorlage für eine Detailseite, auf der ein Blogbeitrag mit seinem vollen Inhalt dargestellt wird, siehe Abbildung 3.

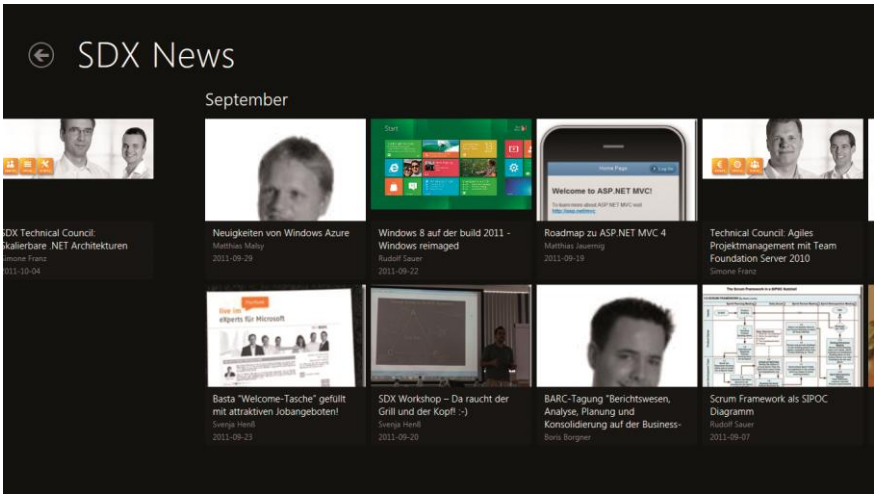



*Abbildung 2*


*Abbildung 3*

## Paketierung und Bereitstellung

Windows Apps werden nicht wie klassische Desktop-Applikationen vertrieben, indem sie von einem beliebigen Medium installiert oder ausgeführt werden. Stattdessen werden sie als Deployment-Paket in einer APPX-Datei paketiert. Technisch gesehen ist es eine ZIP-Datei, was vergleichbar ist mit dem XAP-Format von Silverlight. Dieses Paket muss alle Dateien enthalten, die zur Ausführung der App benötigt werden. Der umstrittene Global Assembly Cache (GAC) gehört damit für Windows Apps der Vergangenheit an.

Das Deployment-Paket kann im neuen Windows Store bereitgestellt werden. Dabei handelt es sich um einen durchsuchbaren Katalog aus Apps, vergleichbar mit dem Windows Phone Marketplace. Für private Endanwender ist dies der einzige Weg, um Windows Apps zu installieren.

Vor der Veröffentlichung einer Windows  App durchläuft diese einen Zertifizierungsprozess bei Microsoft. Im SDK von Windows 8 ist ein „App Certification Kit" enthalten, mit dem Entwickler vorab prüfen können, ob ihre App alle technischen und inhaltlichen Restriktionen einhält.

Der Windows Store wird mit Windows 8 eröffnet und wurde mit den Erfahrungen des Windows Phone Marketplace entwickelt. Er hebt sich durch einige Alleinstellungsmerkmale von der Konkurrenz ab. So können Entwickler neben freien und bezahlten Anwendungen auch Trial- Versionen ihrer Apps anbieten. Dabei können Anwender eine App entweder eine gewisse Zeitspanne oder mit einem eingeschränkten Feature-Set testen, bevor sie sie kaufen müssen. Zudem können Entwickler bei In-App-Purchases individuell die Form der Bezahlung vorgeben, Microsoft schränkt sie hierbei nicht ein.

Auch an die Bedürfnisse von Unternehmen wurde gedacht. So kann der Zugriff auf den Windows Store von Unternehmensrechnern aus über Group Policies eingeschränkt werden. Zudem ist es für Unternehmen möglich, Windows Apps direkt auf Rechner zu deployen und dabei die Store-Infrastruktur zu umgehen.

Weitere Eigenschaften des Stores sind unter anderem die Möglichkeit der Bereitstellung von Open-Source-Apps und das Bewerben von klassischen Win32-Applikationen, für die Links zu externen Seiten angeboten werden können. Weitere Informationen bietet [6].

Das App-Modell von Windows 8 bringt für Entwickler viele Neuerungen mit sich. Neben dem Programmiermodell an sich ist es vor allem das Ausführungsmodell von Windows Apps, für das insbesondere Entwickler klassischer Desktop-Applikationen umdenken müssen.

Allgemein wird eine Windows App in einer Art Sandbox („App Container") mit eingeschränkten Rechten ausgeführt, ähnlich wie mit Silverlight und Silverlight für Windows Phone. Die App hat keinen Zugriff auf systemkritische Ressourcen und nur eingeschränkten Zugriff auf geschützte Bereiche des Benutzers. Das macht sich unter anderem beim Dateisystemzugriff bemerkbar. Eine Windows App hat keinen vollen Zugriff auf das Dateisystem.
Jede Windows App verfügt hingegen über einen abgegrenzten Speicherbereich in den Anwendungsdaten des Rechners [7], auf den nur sie Zugriff hat. Dabei stehen mehrere Datenspeicher zur Verfügung, die im Kasten Datenspeicher für Windows Apps aufgeführt sind. Möchte die App

Zugriff auf weitere Dateien erlangen, so muss dies zum Beispiel über einen File Picker durch den Anwender geschehen.

**Datenspeicher für Windows Apps**

Für die Ablage von Daten stehen einer Windows App drei Arten von Datenspeichern zur Verfügung:

- Local: Speicherbereich für Daten, die nur auf dem lokalen System vorgehalten werden.
- Temporary: Speicherbereich für temporäre Daten. Dieser kann jederzeit vom System geleert werden.
- Roaming: Speicherbereich für Daten, die sowohl lokal als auch in der Cloud (Windows Live) vorgehalten werden. Sie stehen somit auf einem anderen Windows-8-Rechner zur Verfügung, wenn sich ein User mit seiner Live ID an diesem anmeldet. Um die Synchronisation mit der Cloud kümmert sich Windows, für Entwickler fällt keine zusätzliche Arbeit an.

Der Entwickler kann eine Windows App zum Zugriff auf spezielle Ordner berechtigen, wie etwa die Dokumentenbibliothek, Bilder, Videos oder Musik des Benutzers. Dies betrifft auch den Zugriff auf andere Hardware-Ressourcen wie die Datenverbindung oder Sensoren (Lokalisation, Webcam, NFC …). Der Entwickler hinter- legt dazu das benötigte Recht in den App Capabilities der Windows App. Diese lassen sich in Visual Studio in jedem Windows App Projekt über das App-Manifest mit dem Namen Package.appxmanifest konfigurieren, siehe Abbildung 4. So braucht die SDX-News-App beispielsweise die Internet (Client)-Capability, um den RSS-Feed des Blogs abzurufen.

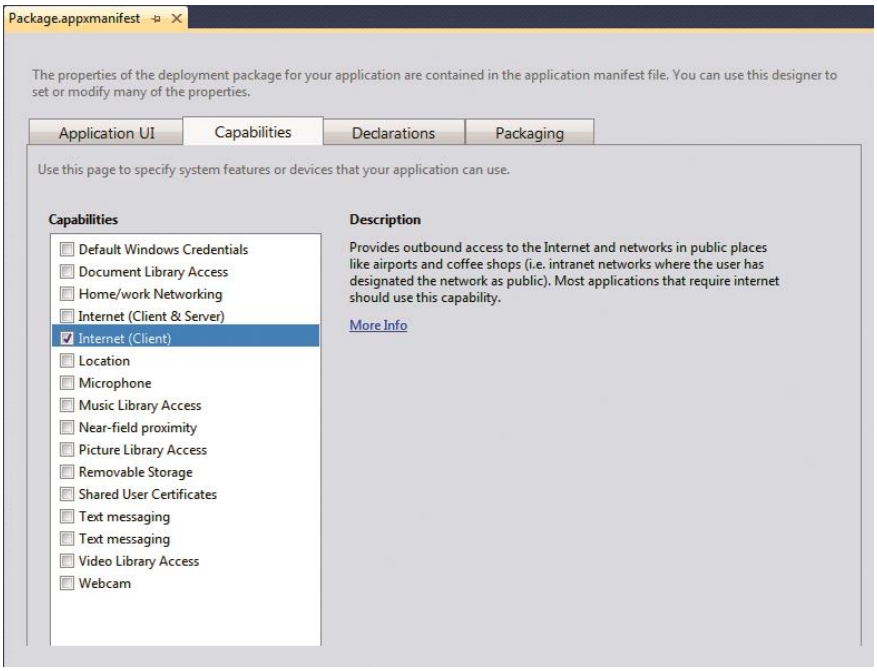



*Abbildung 4*

Die Erlaubnis für eine Capability wird vom Benutzer bei Installation einer Windows App aus dem Windows Store erfragt. Lehnt er ab, so wird der Installationsprozess abgebrochen. Hat ein Entwickler vergessen, eine Capability einzutragen, so wird zur Laufzeit eine Exception ausgelöst oder die App scheitert bereits an der Zertifizierung für den Windows Store.

## Nutzung der WinRT

Die Funktionalität der WinRT lässt sich aus allen unterstützten Sprachen heraus nutzen. Bei der Umsetzung der SDX- News-App mit XAML/C# stellt sich die Nutzung des WinRT-API wie die Verwendung einer ganz normalen .NET-Klassenbibliothek dar.

Ein Beispiel dafür ist in Listing 1 zu sehen. Hier wird für die SDX-News-App der RSS-Feed des Blogs abgerufen, wobei die Klasse SyndicationClient aus dem WinRT- Namespace Windows.Web.Syndication zum Einsatz kommt. Die natürliche Nutzung der WinRT aus den unterstützten Sprachen war eines der Designziele bei der Entwicklung der Bibliothek. Dieses Ziel wurde durch eine „Language Projection" realisiert, die WinRT-Typen und -Konzepte transparent auf die jeweilige Sprache abbildet. Mechanismen wie P/Invoke, die zum Aufruf des Win32-API nötig waren, sind damit bei WinRT obsolet.

**Listing 1**

*Einen Feed mit WinRT aus C# abrufen.*

```
public async Task<IList<NewsItem>> GetNewsAsync() {
        var newsItems = new List<NewsItem>();
```

```
        var client = new SyndicationClient();
        SyndicationFeed feed = await client.RetrieveFeedAsync
                ("http://flurfunk.sdx-ag.de/feeds/posts/default?alt=rss");
        foreach (SyndicationItem item in feed.Items) {
            var newsItem = new NewsItem {
                Title = item.Title.Text,
                PubDate = item.PublishedDate.DateTime
            };
            ...
            newsItems.Add(newsItem);
        }
        return newsItems;
}
```

Die Funktionalität der WinRT-Klassenbibliothek wird an C#-Windows Apps über den Namespace Windows.* exponiert. WinRT kann in mehrere logische Blöcke unterteilt werden, siehe Abbildung 5. WinRT bietet eine hohe Funktionalität, die Sie zur Entwicklung von Windows Apps verwenden können. Neben neuer Funktionalität für Windows Apps, zum Beispiel Tiles, Geolocation, Sensoren und andere, enthält WinRT auch einige Komponenten, die in ähnlicher Form bereits in .NET implementiert sind, wie etwa für die Bereiche Threading/Timers, Cryptography, XML und Networking. Im Vergleich zu .NET als Managed-Code-Framework auf höherer Ebene handelt es sich dabei aber um native Implementierungen. Sie sind im Kern des Betriebssystems verankert, beachten die Konzepte von Windows Apps und stehen allen unterstützten Sprachen zur Verfügung.

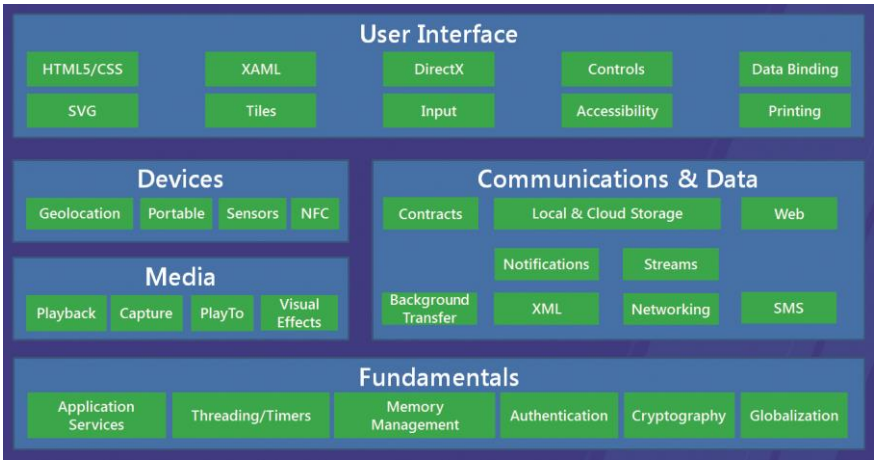



*Abbildung 5*

Zur Entwicklung von Windows Apps mit XAML/C# beziehungsweise XAML/VB steht außerdem das .NET Framework in der neuen Version 4.5 zur Verfügung, allerdings in eingeschränkter Form. Viele

Bestandteile der Base Class Library (BCL) lassen sich zwar nutzen und die Ausführung erfolgt in einer vollständigen .NET 4.5 Runtime. Es kann aber keine Funktionalität verwendet werden, die das Sandboxing-Prinzip von Windows Apps verletzt. Bestandteile wie WPF oder ASP.NET sind ebenfalls außen vor.

Ein eingeschränktes .NET Framework? Das klingt sehr nach Silverlight. Und in der Tat: Auch wenn bei Windows Apps die Silverlight-Technologie nicht mehr unter diesem Namen zum Einsatz kommt, so sind viele Konzepte und Komponenten von Silverlight und vor allem Silverlight für Windows Phone auch in WinRT und Windows Apps verfügbar. Abbildung 6 zeigt das Ergebnis des WinRT Genome Project [8], bei dem die Gemeinsamkeiten von WinRT und Silverlight 5 untersucht wurden. Man sieht, dass es bei den beiden Technologien viele Überschneidungen gibt, WinRT als Plattform-Technologie aber noch viel mehr bietet.

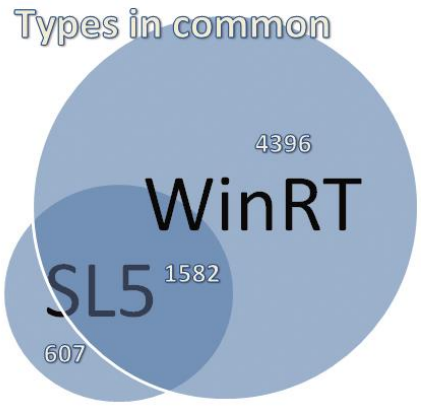



*Abbildung 6*

Insgesamt zeigt sich, dass sich .NET- und vor allem Silverlight-Entwickler bei der Entwicklung von Windows Apps schnell heimisch fühlen werden. Viel vorhandenes Wissen lässt sich übertragen, sodass die wesentliche Umstellung im Erlernen der WinRT und der Design-Aspekte von Windows Apps liegt. Das Prinzip der Asynchronizität von Methodenaufrufen ist vielen Silverlight-Entwicklern bereits vertraut und spielt auch bei der Entwicklung von Windows Apps eine wichtige Rolle. Generell gilt bei WinRT: Für alle Methoden, die länger als 50 ms dauern könnten, ist nur ein asynchrones API verfügbar. Dieses Design-Prinzip von WinRT hilft dabei, dass Windows Apps nicht blockieren, was vor allem bei touchfähigen Apps notwendig erscheint, bei denen eine flüssige Ausführung essenziell ist.

.NET 4.0 hat die asynchrone Entwicklung bereits mit der Task Parallel Library (TPL) und den Basistypen Task und Task<T> stark vereinfacht. C# 5.0 geht nun mit der Einführung der Schlüsselwörter async und

await noch einen Schritt weiter. Damit ist es möglich, Code in prinzipiell sequenzieller Form zu schreiben, der dennoch asynchron ausgeführt wird. Dadurch werden Callback-Methoden und der oftmals damit einhergehende Spaghetti-Code vermieden.

Listing 1 demonstriert auch die Verwendung von async und await. So zeigt das Schlüsselwort await vor dem Aufruf der Methode SyndicationClient.Retrieve- FeedAsync() dem Compiler an, dass eine asynchrone Methode aufgerufen wird. Das Ergebnis dieses asynchronen Aufrufs kann dann wie bei synchronem Code direkt einer Variablen zugewiesen werden. Ohne async/await wäre eine Callback-Methode erforderlich, die bei Beendigung des asynchronen Aufrufs ausgeführt wird.

Mit await wird nicht blockierend auf den Abschluss des asynchronen Aufrufs gewartet, es wird auch kein neuer Thread für den Aufruf gestartet. Vielmehr wandelt der Compiler den entsprechenden Code in ein Task oder Task<T> um, sodass die .NET Runtime damit umzugehen weiß. Dieser Task wird dann normal ausgeführt. Die Umwandlung in einen Task<T> ist auch der Grund, warum die Methode Get-NewsAsync() die Signatur async Task<T> besitzt. Im Code wird zwar T zurückgegeben, durch die interne Compiler-Umwandlung ist der tatsächliche Rückgabewert allerdings ein Task<T>, sodass dies auch in der Methodensignatur enthalten sein muss. Das async-Keyword verdeutlicht Entwicklern, dass die entsprechende Methode asynchron ist und wiederum mit await aufgerufen werden kann. Weitere Informationen zu async/await bietet [9].

## XAML on board

Für die Entwicklung von Windows Apps mit C#, VB und auch C++ steht zur Oberflächengestaltung die aus Silverlight und WPF bekannte Markup-Sprache XAML zur Verfügung. Ebenso lässt sich das bekannte Design-Tool Expression Blend verwenden, das zur Erstellung von Windows Apps in Version 5 bereitsteht. XAML wurde für WinRT nativ implementiert, sodass sich eine erhöhte Geschwindigkeit im Vergleich zu WPF und Silverlight ergibt.

Das XAML für Windows Apps ist von den Fähigkeiten her stark an das XAML von WPF/Silverlight angelehnt. Konzepte wie Data Binding, Event Trigger, Templates, Styling, Visual States und andere können Sie weiterhin verwenden. Auch das MVVM-Pattern lässt sich wie gehabt implementieren. Allerdings müssen Sie die WinRT-Namespaces unterhalb von Windows.* einsetzen. Bei der SDX-News-App kommt zur Darstellung der Blogbeiträge auf der Übersichtsseite beispielsweise eine GridView zum Einsatz, an die eine CollectionViewSource gebunden wird, um die Blogbeiträge in gruppierter Form anzuzeigen. Die Darstellung eines einzelnen Grid-Elements wird über ein DataTemplate beschrieben. Das XAML hierzu liefert Listing 2, das WPF- und Silverlight-Entwickler schnell verstehen werden.

**Listing 2**

*XAML-Code für die GridView auf der Übersichtsseite (Auszug).*

```
<DataTemplate x:Key="BlogPostTemplate">
        <Grid HorizontalAlignment="Stretch">
            ...
            <Border Grid.Column="0">
                <Image Source="{Binding Image}" Stretch="UniformToFill" />
            </Border>
            <StackPanel Grid.Column="1">
                <TextBlock Text="{Binding Title}" />
                <TextBlock Text="{Binding Author}" />
                <TextBlock Text="{Binding PubDate}" />
            </StackPanel>
        </Grid>
</DataTemplate>
...
<GridView x:Name="BlogPostsGridView"
            ItemsSource="{Binding Source={StaticResource BlogPostsDataSource}}"
            ItemTemplate="{StaticResource BlogPostTemplate}" ... />
```

Das WinRT-XAML unterscheidet sich unter anderem durch die Controls. Dazu gehört auch die GridView aus Listing 2. Eine vollständige Liste bietet [10]. Viele dieser Controls sind Touch-optimiert, sodass sie sich im Vergleich zu den Standard-Controls von WPF und Silverlight erheblich besser mit dem Finger bedienen lassen. Dennoch wird die klassische Bedienung per Tastatur/ Maus nicht außer Acht gelassen. So werden zum Beispiel automatisch Scrollbalken eingeblendet, wenn die Bewegung einer Maus erkannt wird und es nötig ist.

Von Windows Phone wurde das Konzept der App Bar entlehnt, einer Art dynamischer Menüleiste. Sie wird entweder permanent eingeblendet oder kommt mit einer Wischgeste vom unteren oder oberen Bildschirmrand zum Vorschein, oder sie wird über einen Klick mit der rechten Maustaste aktiviert, siehe Abbildung 7.


*Abbildung 7*

Die App Bar nimmt Buttons für Aktionen der App auf und kann für jede XAML-Seite individuell gestaltet werden. In der SDX- News-App kann der Anwender beispielsweise über die App Bar von jeder Seite zur Startseite wechseln, siehe Listing 3.

**Listing 3**

*Application Bar mit Home-Button.*

```
<ApplicationBar x:Name="ApplicationBar" VerticalAlignment="Bottom"
                DismissMode="LightDismiss">
    <StackPanel Orientation="Horizontal">
        <Button x:Name="HomeButton" Click="HomeButton_Click"
                Style="{StaticResource AppBarButtonStyle}" />
        <TextBlock Text="Home" Style="{StaticResource AppBarTextStyle}" />
    </StackPanel>
</ApplicationBar>
```

Entwickler von Windows Apps müssen sich viele Gedanken um das UI-Design machen, was Windows-Phone-Entwicklern bekannt sein dürfte. Denn Windows Apps stellen hohe Anforderungen an das Design: Sie dürfen nicht mit Informationen überfrachtet sein, denn die Inhalte brauchen Platz zum Atmen. Entwickler müssen auf Kontextmenüs und überlappende Fenster verzichten und bessere Wege zur Darstellung von Inhalten finden.

Die größte Herausforderung stellt allerdings dar, dass alle Windows Apps im Vollbild ausgeführt werden. Entwickler müssen ihre Apps daher so entwerfen, dass sie auf jeder beliebigen Bildschirmgröße (ab 1024 x 768 px aufwärts) gut aussehen. Die Standard-Controls unterstützen den

Aufbau solch adaptiver UIs durch eine automatische Größenanpassung, dennoch müssen Entwickler dies im Layout ihrer Apps explizit berücksichtigen.

Die Vollbildausführung stellt nicht nur aufgrund beliebiger Auflösungen eine Herausforderung für Entwickler dar. Hat ein Benutzer ein mobiles Gerät, so kann er jederzeit durch Kippen des Geräts zwischen Querformat und Hochformat wechseln, worauf eine Windows App bei Bedarf reagieren sollte. Das ist nicht immer problematisch, so ordnen sich zum Beispiel die Controls bei der SDX-News-App durch den Einsatz eines adaptiven Layouts automatisch neu an, siehe Abbildung 8.

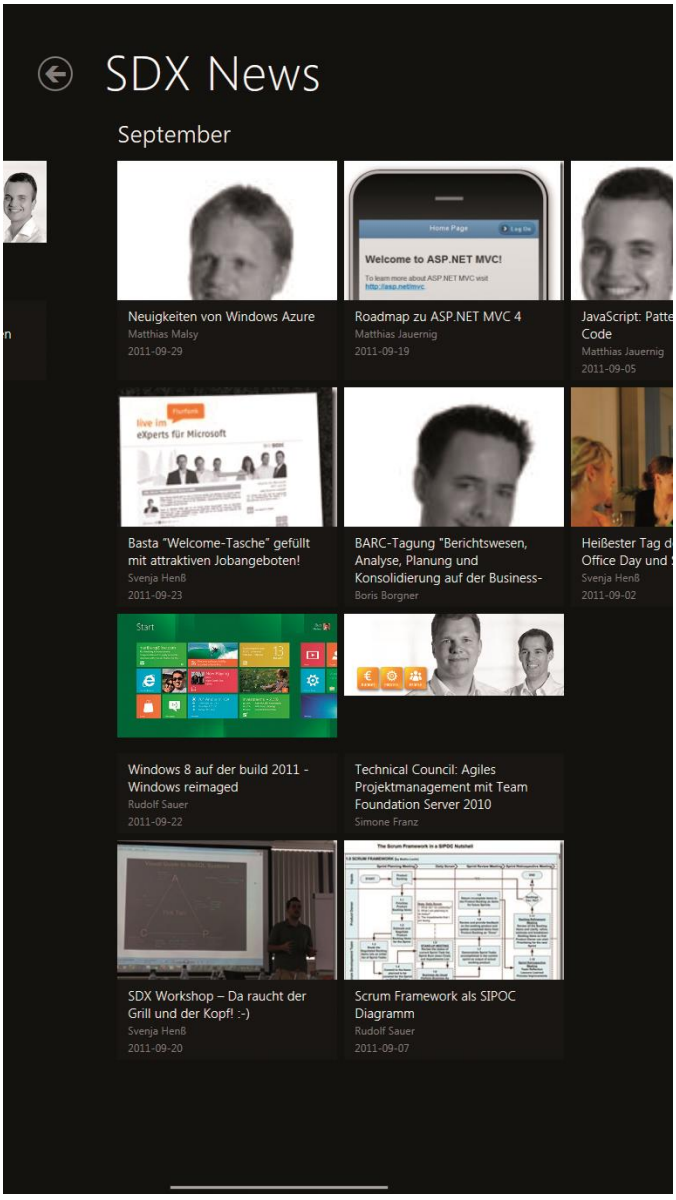



*Abbildung 8*

In anderen Fällen muss allerdings mit einem angepassten Layout auf den Orientierungswechsel reagiert werden. Das lässt sich in XAML elegant mit Visual States lösen und wird im nachfolgenden Abschnitt an einem Beispiel gezeigt.

## Zustandswechsel

Nicht nur auf einen Orientierungswechsel kann man in XAML mit Visual States reagieren, auch ein weiterer Aspekt von Windows Apps ist darüber abbildbar. So stellt die Vollbildausführung zwar den Normalfall dar, wodurch in der Regel immer nur eine App aktiv ist. Windows 8 erlaubt es darüber hinaus aber auch, dass ab einer Bildschirmbreite von 1366  px zwei Anwendungen nebeneinander (side-by-side) ausgeführt werden können. Eine Anwendung nimmt dabei den Hauptteil des Bildschirms ein, während die andere an einer Bildschirmseite angeheftet wird (Snapped State), siehe Abbildung 9. Bei der SDX-News-App kann das zum Beispiel sinnvoll sein, um die Blogbeiträge im kleinen Bereich anzusehen, während im großen Bereich ein Browser zur Anzeige externer Links läuft.

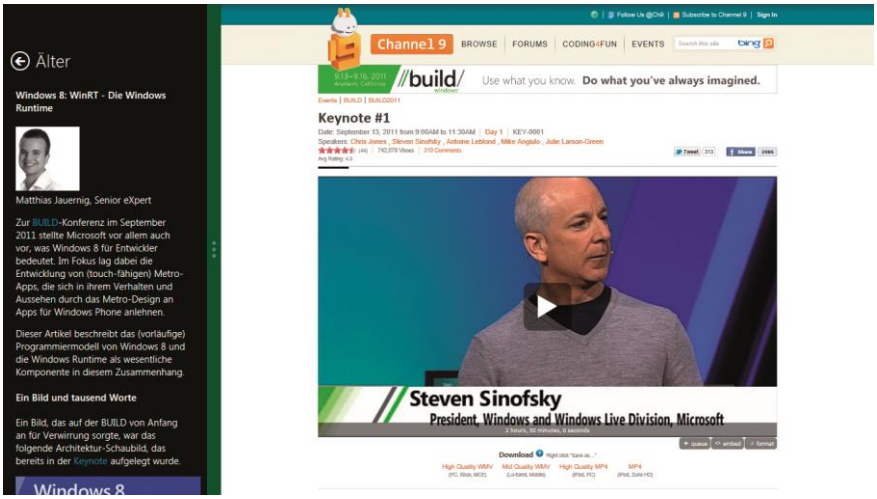



*Abbildung 9*

Auf einen Wechsel in den Snapped State kann ebenso wie auf einen Orientierungswechsel mit einem angepassten Layout reagiert werden, indem dafür ein Visual State in XAML definiert wird. Listing 4 zeigt ein Beispiel für die Definition der Visual States im XAML-Code einer Seite. Dabei wird mit dem Visual State Snapped definiert, dass bei der Ausführung der Windows App an einer Bildschirmseite die Übersicht der Blogbeiträge nicht in einer GridView, sondern in einer ListView angezeigt werden soll. Dies wird durch Setzen der Visibility beider Controls erreicht.

**Listing 4**

*Visual States definieren.*

```
<VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="LayoutStates">
            <VisualState x:Name="Full"/>
            <VisualState x:Name="Fill"/>
            <VisualState x:Name="Portrait" />
```

```
            <VisualState x:Name="Snapped">
                <Storyboard>
                    <ObjectAnimationUsingKeyFrames Storyboard.TargetName="ItemGridView"
                                                   Storyboard.TargetProperty="Visibility">
                        <DiscreteObjectKeyFrame KeyTime="0" Value="Collapsed"/>
                    </ObjectAnimationUsingKeyFrames>
                    <ObjectAnimationUsingKeyFrames Storyboard.TargetName="ItemListView"
                                                   Storyboard.TargetProperty="Visibility">
                        <DiscreteObjectKeyFrame KeyTime="0" Value="Visible"/>
                    </ObjectAnimationUsingKeyFrames>
                </Storyboard>
            </VisualState>
        </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
...
<GridView x:Name="BlogPostsGridView"
        ItemsSource="{Binding Source={StaticResource BlogPostsDataSource}}"
        ItemTemplate="{StaticResource ListItemTemplate}" ... />
<ListView x:Name="BlogPostsListView"
        ItemsSource="{Binding Source={StaticResource BlogPostsDataSource}}"
        ItemTemplate="{StaticResource ListItemTemplate}" Visibility="Collapsed" ... />
```

Was noch fehlt, ist die Verdrahtung der Visual States mit den jeweiligen Ereignissen. Listing 5 zeigt den hierfür notwendigen Code. Beim Laden der Seite werden die Events gebunden, die bei einem Orientierungs- beziehungsweise Layoutwechsel der App ausgelöst werden. Dabei handelt es sich um DisplayProperties. OrientationChanged und ApplicationLayout.GetForCurrentView().LayoutChanged. Über die Methode GetViewState() wird er- mittelt, welcher View State angezeigt wer- den soll, bevor SetCurrentViewState() mit einem Aufruf von VisualStateManager. GoTo State() in diesen wechselt.

**Listing 5**

*Mit Visual States auf Orientierungs- und Layoutwechsel reagieren.*

```
private void Page_Loaded(object sender, RoutedEventArgs e) {
        DisplayProperties.OrientationChanged += s => SetCurrentViewState(this);
        ApplicationLayout.GetForCurrentView().LayoutChanged += (s,e) =>
                        SetCurrentViewState(this);
}
```

```
private void SetCurrentViewState(Control ctrl) {
        VisualStateManager.GoToState(ctrl, GetViewState(), false);
}
private string GetViewState() {
        var orientation = DisplayProperties.CurrentOrientation;
        if (orientation == DisplayOrientations.Portrait ||
            orientation == DisplayOrientations.PortraitFlipped) return "Portrait";
        var layout = ApplicationLayout.Value; if (layout == ApplicationLayoutState.Filled) return "Fill";
        if (layout == ApplicationLayoutState.Snapped) return "Snapped";
return "Full";
}
```

In GetViewState() wird zunächst mit DisplayProperties.CurrentOrientation die Orientierung des Geräts abgefragt. Mit ApplicationLayout.Value wird dann der aktuelle Layout-Zustand der App ermittelt. Dieser Wert vom Enumerationstyp ApplicationLayoutState hat folgende Ausprägungen:

- FullScreen ist gesetzt, wenn die App den kompletten Bildschirmbereich einnimmt (Standard).
- Snapped ist gesetzt, wenn die App side-by-side mit einer anderen Anwendung läuft und den kleineren Bildschirmbereich einnimmt.
- Filled ist gesetzt, wenn die App side-by-side mit einer anderen Anwendung läuft und den größeren Bildschirmbereich einnimmt.

## HTML5/JavaScript

Neben den etablierten Sprachen für Windows-Anwendungen erlaubt Windows 8 auch die Entwicklung von Windows Apps mit HTML und JavaScript. Und so wurde die SDX-News-App zusätzlich zu XAML/C# auch in diesen Sprachen umgesetzt.

Das Ausführungsmodell solcher „Win-WebApps“ ist vollständig clientseitig, das heißt, es findet nicht wie zum Beispiel bei ASP.NET ein serverseitiges Hosting der erstellten App statt. Vielmehr wird aus den Inhalten der App ein normales APPX- Paket erstellt, das bei Ausführung mit der Render-Engine des IE10 dargestellt wird. Der JavaScript-Code wird in der hardwarebeschleunigten Chakra-Engine des IE10 ausgeführt, siehe auch Abbildung 1. Dies ist nicht zu verwechseln mit einer Ausführung im IE10. Stattdessen wird die App wie jede andere Windows App in einem eigenen App-Container ausgeführt und verfügt damit über alle bereits benannten Aspekte einer Windows App. Wie bei normalen Webanwendungen dient auch bei solchen Windows Apps HTML in Kombination mit CSS zur Oberflächengestaltung, während JavaScript zur Steuerung und zur Definition von UI-Logik zum Einsatz

kommt. Für die Gestaltung des UI kann Blend 5 verwendet werden, das neben XAML für Windows Apps auch HTML und CSS unterstützt.

Anders als bei Modellen wie ASP.NET Web Forms oder MVC verzichtet Microsoft bei der Definition von HTML-Seiten auf Custom Markup. Das zum Einsatz kommende HTML ist standardkonform und wird so dargestellt, wie es vom Entwickler definiert wurde. Das heißt, es können auch alle vom IE10 unterstützten Features von HTML5 und CSS3 verwendet werden. Eine Anreicherung des UI mit Windows-spezifischer Funktionalität er- folgt lediglich über die Definition von Attributen der Form data-win-* auf HTML- Elementen. Diese data-Attribute (sprich: Data Dash) sind Bestandteil des HTML5-Standards und kommen beispielsweise auch bei den JavaScript-Bibliotheken jQuery oder KnockoutJS zum Einsatz. Bei der SDX-News-App wird auf der Startseite zum Beispiel mit dem Attribut-Term da- ta-win-control="WinJS.UI.ListView" auf einem div definiert, dass sich das div als Liste von Elementen rendern soll, um die Übersicht der Blogbeiträge darzustellen.

Dieses Beispiel nutzt auch die Java- Script-Bibliothek WinJS [11], die Bestand- teil jedes HTML/JS-Windows App Projekts ist und die eigentliche Funktionalität von Windows Apps bereitstellt. Diese Bibliothek bein- haltet neben Touch-optimierten Controls und CSS-Styles auch Hilfsfunktionalität für die Erstellung von Windows Apps mit JavaScript. WinJS und WinRT sind auch dafür verantwortlich, dass mit HTML/JavaScript erstellte Windows Apps nicht platt- formunabhängig sind. Zwar lassen sich Apps auch ohne diese Bibliotheken erstellen, doch verlieren sie dann alle Features, die eine Windows App ausmachen. Dennoch sollte man das HTML-JavaScript-basierte Programmiermodell von Windows 8 nicht unterschätzen, erlaubt es doch die Verwendung beliebiger bekannter JavaScript- Bibliotheken wie zum Beispiel jQuery, deren Funktionalität sich zur Erstellung von Windows Apps nutzen lässt. Microsoft ermöglicht damit der großen Masse von Webentwicklern einen schnellen Einstieg in die Entwicklung von Apps unter Windows 8.

## Ein Template, sie zu binden

In Listing 6 ist ein Ausschnitt des HTML-Markups von der Startseite der SDX-News-App zu sehen, die als Übersichtsseite für die abgerufenen Blogbeiträge dient. Das div namens postsList bildet den Container für die darzustellenden Blogbeiträge. Zudem wird im div namens itemTemplate festgelegt, wie ein einzelner Blogbeitrag in der Übersicht dargestellt werden soll. Dazu wird es über das Attribut data-win-control als Template-Control definiert. Somit wird es nicht wie ein normales HTML-Element angezeigt, sondern dient als Vorlage für Elemente einer Aufzählung, ähnlich einem Template in XAML, vergleiche dazu Listing 6 mit Listing 2. Innerhalb des Templates können dann mit data-win-bind

Properties der zugeordneten Entität an HTML-Elemente gebunden werden. Die Bindung kann an ein beliebiges Attribut eines  HTML-Elements erfolgen.

**Listing 6**

*HTML-Markup der Startseite der SDX-News-App (Auszug).*

```
 <div id="itemTemplate" data-win-control="WinJS.Binding.Template">
        <div class="largeTileTextTemplate">
            <div><img style="width:100%" alt="Vorschaubild" data-win-bind="src: image" />
            </div>
                <div class="largeTileTextTemplateOverlay">
                    <div data-win-bind="textContent: title"></div>
                    <div data-win-bind="textContent: author"></div><
                    div data-win-bind="textContent: pubDate"></div>
                </div>
        </div>
</div>
...
<div id="postsList" data-win-control="WinJS.UI.ListView"
        data-win-options="{
            dataSource: blogData.posts,
            itemRenderer: itemTemplate,
            layout: {type: WinJS.UI.GridLayout, groupHeaderPosition: 'top'},
             ...}">
</div>
```

Über das Attribut data-win-options kann das Verhalten der ListView festgelegt werden. So ist es mit dataSource möglich, das JavaScript-Objekt anzugeben, das als Datenquelle dienen soll (bei blogData. posts handelt es sich um die zuvor abgerufenen Blogbeiträge). Weiterhin kann mit itemRenderer das Template angegeben werden, das zur Darstellung der einzelnen Listenelemente verwendet werden soll, und so weiter.

Insgesamt lassen sich viele Aktionen zum Aufbau des UI bereits über das HTML- Markup erledigen, ohne dass der Entwickler auf JavaScript zurückgreifen muss, was bei XAML ähnlich ist. JavaScript kommt dann zum Einsatz, wenn es um das eigentliche Abrufen von Daten und weitere UI-Logik geht, die nicht deklarativ angegeben werden kann. So kann man zum Beispiel auch in JavaScript auf

Orientierungs- oder Layoutwechsel reagieren, indem man die Darstellung der App entsprechend anpasst.

Ebenso wie aus den anderen unterstützten Sprachen lässt sich die komplette Funktionalität der Windows Runtime zur Erstellung von Windows Apps auch mit JavaScript nutzen. Bei der SDX-News-App lässt sich zum Beispiel mit dem JavaScript- Code aus Listing 7 der RSS-Feed des Blogs abrufen, ähnlich wie aus C# in Listing 1. Die zugrunde liegende Funktionalität der WinRT ist dieselbe. Interessant ist der Aufruf der asynchronen Methode SyndicationClient.retrieveFeedAsync(). In C# 5 kann diese asynchrone Methode elegant mit dem neuen Schlüsselwort await aufgerufen werden. In JavaScript setzt Microsoft hingegen auf das Konzept der „Promises", das bereits aus JavaScript-Frameworks wie jQuery oder Dojo bekannt ist. In Listing 7 spiegelt sich das in der Funktion then() wider. Dieser Funktion kann als erster Parameter eine Funktion übergeben werden, die aufgerufen wird, wenn der asynchrone Aufruf erfolgreich zurückkehrt. Als weitere Parameter können Funktionen angegeben werden, die bei einem Fehler oder bei einer Änderung des Fortschritts der Ausführung ausgeführt werden. Promises ermöglichen ein standardisiertes API für asynchrone Aufrufe. Wer Promises für eigenen Code nutzen möchte, kann dies über die Funktion WinJS.Promise() tun.

**Listing 7**

*Asynchroner Abruf eines Feeds mit WinRT aus JavaScript.*

```
function getNewsAsync(callback) {
        var client = new Windows.Web.Syndication.SyndicationClient();
        client.retrieveFeedAsync(new Windows.Foundation.Uri
                ("http://flurfunk.sdx-ag.de/feeds/posts/default?alt=rss"))
            .then(function (feed) {
                var newsItems = [];
                for (var i = 0; i < feed.items.length; i++) {
                    var item = feed.items[i];
                    var newsItem = new NewsItem();
                    newsItem.title = item.title.text;
                    newsItem.pubDate = item.publishedDate;
                    ...
                    newsItems.push(newsItem);
                }
                callback(newsItems);
        });
}
```

## Eigene WinRT-Komponenten

WinRT ermöglicht darüber hinaus noch ein ganz anderes Szenario. So ist es möglich, eigene WinRT-Komponenten in C#, VB oder C++ zu schreiben, die dann aus den anderen unterstützten Sprachen her- aus verwendet werden können. So lässt sich etwa eine WinRT-Komponente in C# erstellen, die dann aus JavaScript genutzt werden kann. Das ermöglicht es beispielsweise, Framework-Funktionalität oder einen Business-Layer in C# zu entwickeln, der dann von einem HTML/JavaScript-UI verwendet werden kann. Die Nutzung gestaltet sich genauso natürlich wie bei je- dem anderen WinRT-API auch. Zur Erstellung einer eigenen WinRT-Komponente in C#/VB fügen Sie in Visual Studio der Solution ein neues (leeres) Projekt hinzu. Setzen Sie den Output type dieses Projekts dann auf WinMD file. Bei WinMD (Windows Metadata) handelt es sich um das Metadatenformat von WinRT, das sich WinRT-Komponenten mit .NET- Komponenten teilen.

Neben den Vorteilen eigener WinRT- Komponenten gibt es allerdings auch einige Einschränkungen, auf die man als Entwickler achten sollte. Diese Einschränkungen beziehen sich auf die öffentlichen Member des Komponenten-API, das heißt auf alle Bestandteile, die für andere Komponenten sichtbar sind. Die zu beachtenden Bedingungen werden im Kasten Bedingungen an eigene WinRT-Komponenten dargestellt, weitere Informationen finden sich unter [12]. Trotz dieser Einschränkungen ist die Erstellung von eigenen WinRT-Komponenten eine reizvolle Möglichkeit, um sprachenübergreifende Funktionalität bereitzustellen.

**Bedingungen an eigene WinRT-Komponenten**

Eigene WinRT-Komponenten in C# müssen folgende Bedingungen erfüllen [12]:

- API-Signaturen dürfen nur WinRT-Typen und selbst definierte Typen verwenden (bzw. .NET-Typen, wenn diese transparent auf WinRT-Typen mappen, wie z.B. IList<T>).
- Exponierte Klassen müssen sealed sein.
- Structs dürfen nur öffentliche Felder beinhalten.
- Es werden nur vordefinierte generische Typen unterstützt, das öffentliche API darf keine eigenen generischen Typen enthalten.

Weitere Windows Aspekte Windows 8 bietet Entwicklern neben den vorgestellten Möglichkeiten noch einige weitere Features an, die sie zur Entwicklung von Windows Apps verwenden können und die in diesem Artikel zumindest erwähnt werden sollen.

Viele dieser Features betreffen eine stärkere Integration der App in das Windows UI. Dazu zählen unter anderem Live Tiles, über die grundsätzlich erst einmal Apps vom Windows UI aus gestartet werden können. Im Gegensatz zu normalen Icons können diese Tiles allerdings auch Informationen der App anzeigen. So wäre es bei einer News-App beispielsweise möglich, dass eine Vorschau zu den neuesten Blogbeiträgen auf dem Live Tile angezeigt wird. Das erlaubt die Darstellung relevanter Inhalte, ohne dass die App eigentlich gestartet ist. Außerdem ist es möglich, tief verlinkte In- halte einer App als Live Tile auf dem Startscreen abzulegen und beim Start der App direkt zum jeweiligen Inhalt zu gelangen. Ein weiteres wichtiges Konzept sind die sogenannten „Contracts". Durch deren Implementierung lassen sich App-übergreifende Aktionen durchführen, ohne dass sich die Apps gegenseitig kennen müssen und ohne dass der Kontext der aktuellen App verlassen werden muss. Sie dienen Windows Apps damit zur Integration in die Windows-Landschaft. Der Kasten App Contracts listet auf, welche Contracts von Entwicklern implementiert werden können. Weitere Informationen bietet [13]. Aus Benutzersicht lassen sich viele dieser Contracts über die sogenannten „Charms" ansprechen. Diese erscheinen durch eine Wischgeste vom rechten Bildschirmrand, siehe Abbildung 10.

**App Contracts**

Ein Entwickler kann unter anderem folgende Contracts implementieren, um seine App stärker in das Windows App UI zu integrieren:

- Search erlaubt die Suche nach Inhalten in einer App oder auf der aktuellen Ansicht der App. Windows Apps können sich auch in die systemweite Suche integrieren, sodass in ihnen nach Inhalten gesucht werden kann, selbst wenn die App geschlossen ist.
- Share erlaubt den Austausch von Daten einer App mit anderen Apps, zum Beispiel Texte, URIs, HTML, Bilder, aber auch selbst definierte Formate. Ein Beispiel hierfür wäre ein Link, der von einem News-Reader an eine Facebook-App weitergegeben wird, um als Status-Update zu erscheinen.
- App to App Picking erlaubt das Auswählen von Dateien/Inhalten aus einer anderen App heraus. Zum Beispiel könnte eine App ein Foto aus einer Flickr-App auswählen, wenn diese den Picker-Contract implementiert.
- Settings und Play To sind weitere Contracts, die Entwickler implementieren können. Eine Übersicht hierzu und weitere Informationen liefert [13].

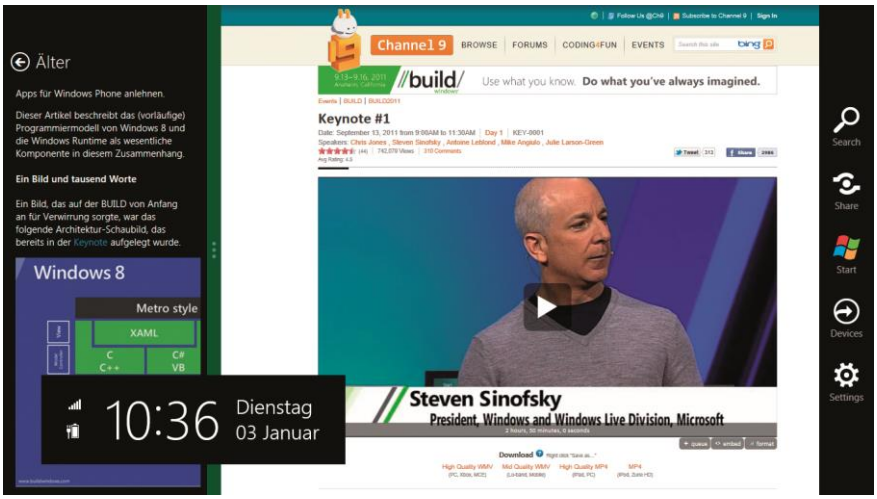



*Abbildung 10*

## Multitasking

Eine weitere wichtige Eigenschaft von Windows Apps ist der Umgang mit Multitasking. Auch hier müssen Entwickler klassischer Desktop-Applikationen umdenken. Denn sobald eine im Vollbildmodus ausgeführte Windows App nicht mehr sichtbar ist, wird sie in den „Suspended State" überführt. Ihre Inhalte verbleiben im Arbeitsspeicher, sodass sie bei erneuter Aktivierung direkt zur Verfügung steht. Allerdings wird die App im Suspended State nicht mehr aktiv ausgeführt. Sie verbraucht damit keine CPU-Zyklen mehr, was dem Ziel eines ressourcenschonenden Betriebssystems zugutekommt.

Windows Apps können auf die Unterbrechung oder Reaktivierung bei Bedarf reagieren. Zudem können definierte Prozesse auch im Hintergrund weiter ausgeführt werden, zum Beispiel ein Musik-Player mit dem „Playback Manager" sowie Download- und Upload-Prozesse von Dateien. Zudem lassen sich über „Background Tasks" [14] Aktionen definieren, die bei Auftreten bestimmter Ereignisse ausgeführt werden, etwa wenn eine Zeitspanne verstrichen ist oder bei Auftreten eines System- oder Netzwerkereignisses [15].

## Fazit

Dieser Artikel bot einen ersten Einblick in das Programmiermodell von Windows 8 und die Erstellung von Windows Apps. Das neue Konzept der Windows Apps stellt in Kombination mit dem Windows Store ein großes Potenzial für Entwickler dar, die mit der neuen Windows Runtime eine hohe Funktionalität zur Erstellung ihrer Apps geboten bekommen. Gleichzeitig müssen bisherige Windows-Entwickler erst einmal umdenken, denn viele Aspekte von Windows  Apps entsprechen nicht der Entwicklung klassischer Desktop-Applikationen.

.NET-Entwickler können ihr Wissen mit in die Erstellung von Windows Apps einbringen, von Vorteil sind Erfahrungen mit XAML, Silverlight und/oder Windows Phone. Sie können die grundlegende .NET-Funktionalität nutzen und es ist wahrscheinlich, dass bald auch bekannte Frameworks und Komponentenbibliotheken auf .NET-Basis zur Entwicklung von Windows Apps zur Verfügung stehen. Das HTML/JavaScript-Programmiermodell steht dem in nichts nach. Durch die Nutzung von WinJS und der WinRT ergibt sich eine hohe Funktionalität, zudem können Entwickler bekannte JavaScript-Bibliotheken wie jQuery einsetzen. Da Windows Apps auf HTML/JavaScript-Basis nicht plattformunabhängig sind, ist die Wahl zwischen XAML/C#, HTML/JavaScript und den anderen Sprachen am ehesten eine Frage des eigenen Wissens und der Anforderungen an die zu erstellende App.

Entwickler bekommen mit dem Programmiermodell von Windows 8 eine weitreichende Flexibilität bei der Erstellung von Windows Apps geboten, die so auf keiner anderen App-Plattform existiert. Die Windows Runtime mit ihrem einfach verwendbaren API und ihrer umfassenden Funktionalität bildet ein breites Fundament für die Erstellung hervorragender Windows Apps.


[1] Webseite der BUILD-Konferenz, www.buildwindows.com

[2] Holger Schwichtenberg, COM doch, dotnetpro 11/2011, Seite 20ff., www.dotnetpro.de/A1111DVCS

[3] Windows 8 Developer Preview, www.dotnetpro.de/SL1203Win8App1

[4] Windows 8 Development Platform Clarified, www.dotnetpro.de/SL1203Win8App2

[5] SDX Flurfunk, http://flurfunk.sdx-ag.de

[6] Previewing the Windows Store, www.dotnetpro.de/SL1203Win8App3

[7] Application Data Overview, www.dotnetpro.de/SL1203Win8App4

[8] WinRT Genome Project, www.dotnetpro.de/SL1203Win8App5

[9] Async Whitepaper, www.dotnetpro.de/SL1203Win8App6

[10] WinRT XAML Controls List, www.dotnetpro.de/SL1203Win8App7

[11] Windows Library for JavaScript reference, www.dotnetpro.de/SL1203Win8App8

[12] Using WinRT from C# and VB, www.dotnetpro.de/SL1203Win8App9

[13] Windows application contracts, www.dotnetpro.de/SL1203Win8App10

[14] Introduction to Background Tasks, www.dotnetpro.de/SL1203Win8App11

[15] Metro App Realtime communication, www.dotnetpro.de/SL1203Win8App12



Mit freundlicher Unterstützung von:

dotnetpro

www.dotnetpro.de

# Kapitel 8: Cloud Development

## Peter Kirchner (Microsoft) – Cloud Computing für jede Anforderung mit Windows Azure


### Arbeitgeber

• Microsoft Deutschland GmbH

### Technologieschwerpunkte

• Technical Evangelist

• Cloud Computing mit Windows Azure

### Mein Urlaubstipp

Mein Lieblingsurlaubsziel sind die Strände im Osten von Fuerteventura! Schöne weiße Strände, hohe Wellen kombiniert mit feinem Sand!

### Meine Inseltechnologie

Cloud Computing ist derzeit mein absoluter Favorit! Als Entwickler war ich es gewöhnt, häufig Test-Systeme für alle möglichen Zwecke aufzusetzen und zu konfigurieren – sehr häufig in virtuellen Maschinen. Jetzt kann ich dies mit jedem PC in der Cloud machen und wenn ich anderen Leuten, die Test-Systeme zeigen möchte, muss ich keine schweren Maschinen mit mir herumtragen.

## Cloud Computing für jede Anforderung mit Windows Azure

Zu Beginn möchte ich immer ein gemeinsames Verständnis dafür schaffen, was wir unter Cloud Computing verstehen. Der Begriff wird mittlerweile häufig und in verschiedenen Interpretationen

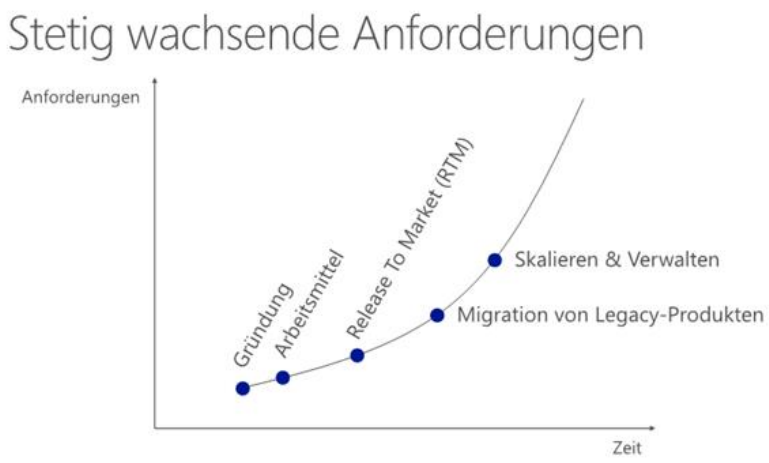
verwandt. Folgende sieben Wesensmerkmale der Cloud sind meiner Meinung nach wichtig, wenn wir über Cloud-Dienste sprechen.


1. Die Cloud steht in praktisch unbeschränkter Menge zur Verfügung.
2. Ressourcen in der Cloud können bedarfsgerecht genutzt werden. D.h. ich kann Ressourcen flexibel und mit sehr kurzer Ankündigung nutzen und auch wieder freigeben.
3. Ressourcen in der Cloud sind vertikal oder horizontal skalierbar.
   - o Vertikale Skalierung bedeutet, dass Ressourcen durch Änderung ihrer Spezifikationen skaliert werden. Bei virtuellen Maschinen ist dies typischerweise die Anzahl der CPUs, die Größe des RAMs oder die Bandbreite des angeschlossenen Netzwerks.
   - o Horizontale Skalierung bedeutet, dass die Anzahl der Ressourcen variiert wird. Um beim Beispiel der virtuellen Maschinen zu bleiben, bedeutet dies hier, mehr oder weniger virtuelle Maschinen für den gleichen Dienst einzusetzen.
4. Ressourcen in der Cloud werden nutzungsabhängig abgerechnet. Es wird nur die Leistung in Rechnung gestellt, die auch genutzt wurde.
5. Ressourcen in der Cloud können über standardisierte Schnittstellen kontrolliert und gesteuert werden.
6. Ressourcen weisen eine garantierte Hochverfügbarkeit auf, die vertraglich vereinbart wurde.
7. Die verschiedenen Angebote können einzeln oder kombiniert eingesetzt werden und man ist nicht auf ein bestimmtes System eingeschränkt.

**Viele dieser Eigenschaften sind wichtig, um als Cloud-Dienst zu bestehen.** Dadurch lassen sich viele bereits existierende Dienste auch als Cloud-Dienste identifizieren, aber auch etliche nicht, die allerdings im momentanen Cloud-Hype trotzdem den Stempel Cloud erhalten.
Im Folgenden betrachten wir beispielhaft den Weg von der Gründung einer Firma bzw. der Idee für eine Lösung bis hin zu dem Zustand, dass die entworfene Lösung erfolgreich am Markt angekommen ist. Dabei hebe ich neben der Gründung vier weitere Zeitpunkte hervor:

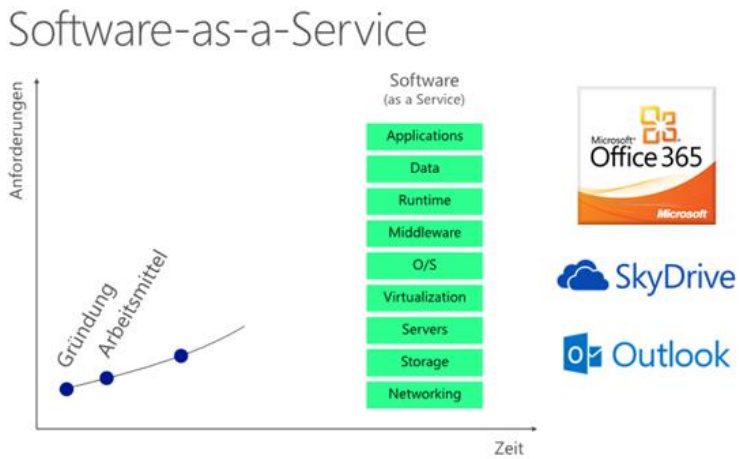
- Beschaffung der Arbeitsmittel fürs Team
- Entwicklung der Lösung (die eigentliche Idee)
- Einbeziehen von älterer Software
- Skalieren & Verwalten der Lösung

Anhand dieser vier Phasen lassen sich die verschiedenen Konzepte der Cloud gut veranschaulichen.




## Beschaffung der Arbeitsmittel fürs Team

Zu Beginn braucht das Team die Arbeitsmittel für die Entwicklung, die Kommunikation und die Kollaboration. Viele dieser Arbeitsmittel lassen sich bereits durch Cloud-Dienste abbilden. Für die Kommunikation kann man hier typischerweise E-Mail-Dienste sehen. Beispiele aus dem Portfolio von Microsoft für private Kunden sind hier Outlook.com und für gewerbliche Kunden Office 365. In Office 365 sind natürlich wesentlich mehr Dienste enthalten als die reine Kommunikation per E-Mail. Durch Lync, Exchange und SharePoint sind hier auch die Team-Kollaboration mit abgedeckt.

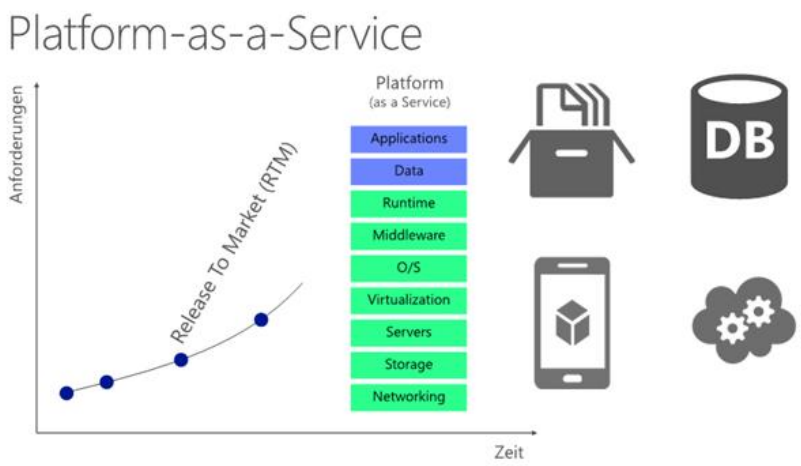



Diese Dienste sind als Software-as-a-Service (SaaS) zu klassifizieren, da hier der Nutzer des Dienstes den Dienst konsumieren kann, ohne sich um die tieferliegenden Schichten des Verwaltungsstack zu

kümmern. Die Beschaffung und Wartung der notwendigen Server, die Installation, Konfiguration und Pflege der Betriebssysteme und die Installation und Pflege der Softwarelösung übernimmt der Dienstanbieter (in der Grafik sind die vom Cloud-Anbieter verwalteten Komponenten in grün dargestellt, was bei SaaS dem gesamten Stack entspricht).

## Entwicklung der Lösung

Die Entwicklung der Lösung, für die man eine Idee hat, ist die wichtigste Phase auf dem Weg zum Vertrieb. Bei der Neuentwicklung einer Lösung hat man die Chance, bereits wichtige Entscheidungen zu treffen, um vornherein die Architektur der Lösung für die Cloud vorzubereiten.

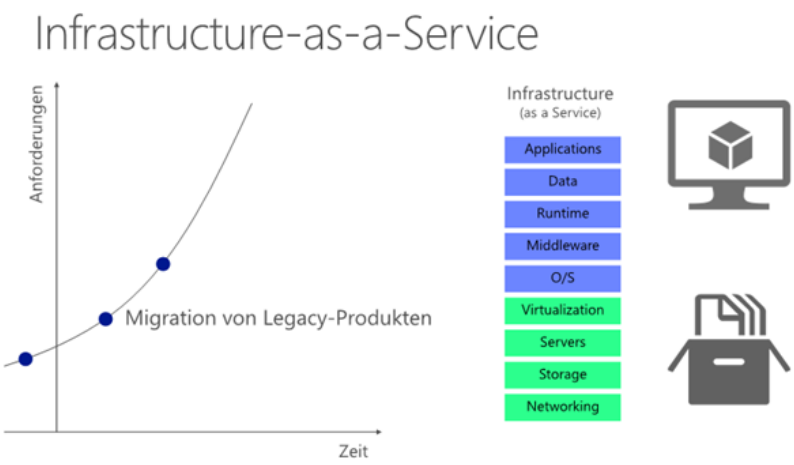



Eine Lösung direkt als Cloud-Lösung zu entwickeln, gibt die Möglichkeit Plattform-as-a-Service (PaaS) zu nutzen. Für Softwarehersteller bedeutet dies, sich auf die Softwarelösung zu fokussieren und die auszuführende Plattform als gegeben vorauszusetzen. Alle Aufgaben für die Pflege der Betriebssysteme und der notwendigen Hardware entfällt somit für den Softwarehersteller und obliegt der Verantwortung des Cloud-Anbieters. (In der Grafik sind die vom Cloud-Anbieter verwalteten Komponenten in grün dargestellt.)

Wichtige Cloud-Ressourcen sind hier unter anderem Speicherplatz, Datenbanken, Cloud-Dienste und (neu) die Windows Azure Mobile Services.

## Einbeziehen von älterer Software

Cloud Computing ist unter anderem auch damit berühmt geworden, dass virtuelle Maschinen nach Bedarf gebucht werden können. Bei dieser generischen Form, Ressourcen in der Cloud zu verwenden, sprechen wir von Infrastructure-as-a-Service (IaaS).



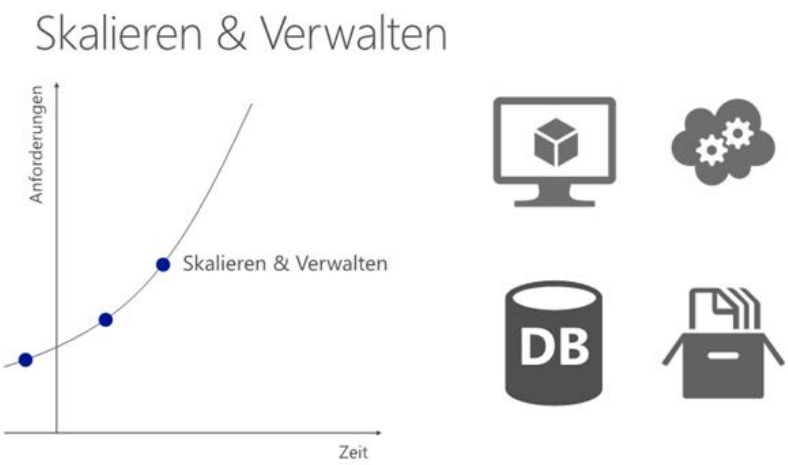

Als Nutzer von IaaS haben wir allerdings nicht nur die größte Freiheit in der Verwendung der Ressourcen, sondern auch die meiste Arbeit. Vom Cloud-Anbieter wird die Beschaffung und Verwaltung der Hardware übernommen. Um die darüber liegenden Schichten muss sich der Nutzer selbst kümmern. (In der Grafik sind die vom Cloud-Anbieter verwalteten Komponenten in grün dargestellt.)

Diese Flexibilität ist dann wichtig, wenn Bestandssoftware in die Cloud gebracht werden soll und nicht angepasst oder modernisiert werden soll oder kann. Auch wenn bestehende PaaS- oder SaaS-Angebote nicht ausreichend sind, bietet IaaS hier die Flexibilität, Serverlösungen nach den eigenen Anforderungen auszurichten.

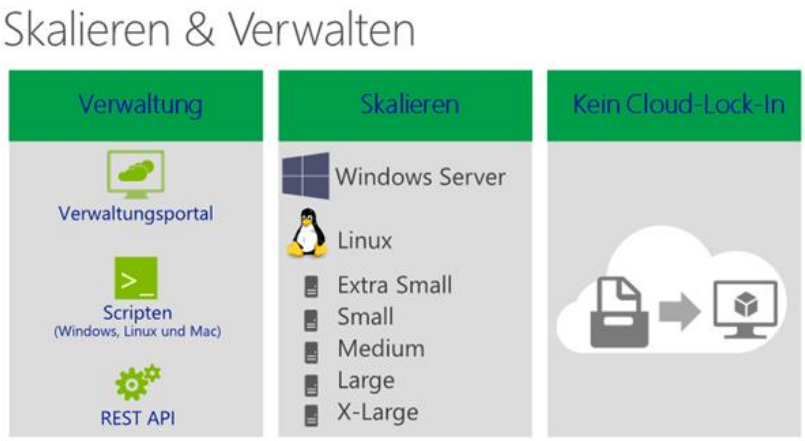
Wichtige Dienste von Windows Azure sind in diesem Zusammenhang die virtuellen Maschinen und natürlich der Speicherplatz, in dem die virtuellen Festplatten abgelegt sind.

## Skalieren & Verwalten der Lösung

Sobald die Lösung erfolgreich läuft, entsteht schnell das Bedürfnis die Lösung effizienter verwalten zu wollen und automatisch zu skalieren. In Windows Azure besteht die Möglichkeit, nahezu alle Ressourcen programmatisch oder per Skript zu verwalten und zu überwachen.




In Windows Azure ist es nicht nur über das Verwaltungsportal möglich Einstellungen von Ressourcen in Windows Azure zu prüfen und zu ändern, sondern auch über PowerShell unter Windows, über Kommandozeilentools unter Windows, Linux und Mac und über REST APIs.




Dazu gehören natürlich auch die Instanzengrößen von virtuellen Maschinen oder von Cloud-Diensten, als auch die Anzahl von Instanzen. Somit ist es möglich hier vertikal als auch horizontal automatisiert zu skalieren.

Ein weiteres wichtiges Merkmal für die Flexibilität ist hierbei ein einheitliches Format für die verwendeten virtuellen Festplatten. Der Hyper Visor vom Windows Server als auch von Windows Azure verwenden das gleiche Format VHD, womit es dem Nutzer möglich ist, die virtuellen Festplatten flexibel entweder im eigenen Rechenzentrum oder in der Cloud in Windows Azure zu verwenden, ohne dass hierbei Konvertierungen zwischen verschiedenen Formaten notwendig sind.

# Impressum

Microsoft Deutschland GmbH

Konrad-Zuse-Straße 1

D-85716 Unterschleißheim

**Verlags-Kontakt:**

O'Reilly Verlag GmbH & Co. KG

Microsoft Press Division

c/o Microsoft Deutschland GmbH

Konrad-Zuse-Straße 1

D-85716 Unterschleißheim

Tel.: +49-89-3176-0

Fax: +49-89-3176-1000

**ISBN:** 978-3-8483-4053-8