Agiles SOA oder was?

Ein zuverlässiger Verfügbarkeitsservice

Dominik Rose, Karl-Heinz Wichert

Bei SOA denkt man nicht unbedingt an Agilität. Dazu passend wird agilen Softwareentwicklern gerne unterstellt, dass sie von Architektur nichts wissen wollen, ob serviceorientiert oder nicht. Wir haben in einem vor kurzem abgeschlossenen Projekt versucht, beides zu verbinden. Im Gegensatz zu vielen (über-)ambitionierten SOA-Projekten hatte es nicht den Anspruch, die Welt zu verändern. Stattdessen löste es ein konkretes Problem im Kerngeschäft des Kunden, an dem vorangegangene Projekte ohne SOA und Agilität gescheitert waren. Wir berichten von dem fachlichen Hintergrund, unserem Vorgehen und den benutzten Technologien. Vor allem stellen wir die Frage, was wir wirklich gemacht haben: Agile Entwicklung, SOA oder sogar beides?

Sonntagmorgen, 5:30 Uhr. Nach acht Stunden ist die Migration endlich erfolgreich abgeschlossen. Jetzt müssen nur noch die ESBs und die Applikationsserver in Betrieb genommen und natürlich der Anwendertest bestanden werden. Dann würde das erste serviceorientierte System des Kunden den Betrieb aufnehmen.

Sonntagnachmittag, 16:00 Uhr. Auch wenn die ESBs sich lange wehrten, ist der Anwendertest bestanden und das System erfolgreich in Betrieb genommen. Mit allen Schwierigkeiten war der Go-Live ein letztes Indiz dafür, dass das Projekt ohne Flexibilität und Agilität niemals möglich gewesen wäre. Doch der Reihe nach …

Das Setting

Die IT-Landschaft des regionalen Telekommunikationsanbieters war historisch gewachsen. Das Auftragsmanagement war eine Eigenentwicklung, die durch die Integration mit weiteren Systemen deutlich an Funktionalität, aber auch an Komplexität gewonnen hatte. CRM- und Billing-System wurden von externen Zulieferern integriert und gepflegt. Der Onlineauftritt war teils selbst gebaut, teils von extern und hatte schon bessere Zeiten gesehen.

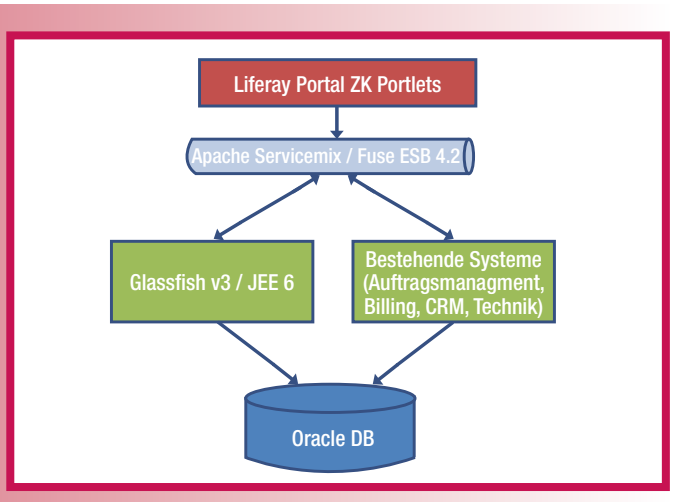

Abb. 1: Schematische Darstellung des Technologie-Stacks

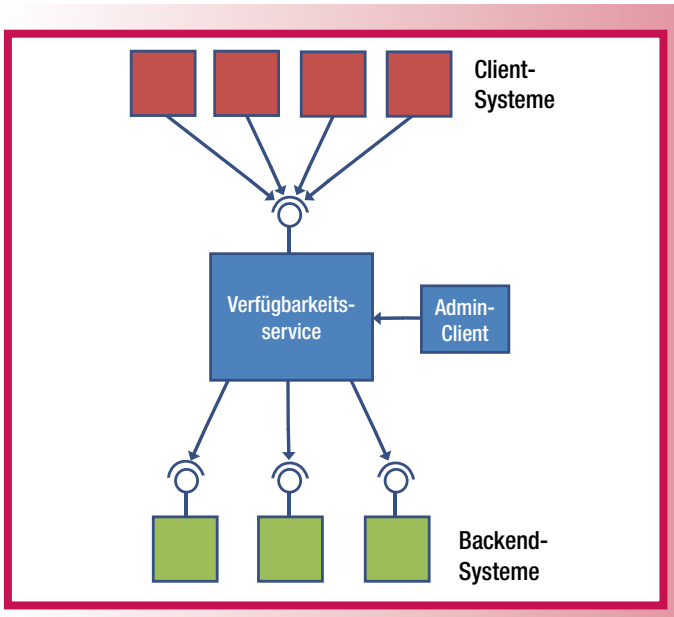

Abb. 2: Zusammenhang der Services und Komponenten. Jeder geschlossene Kreis steht für einen Service. Die Komponenten oben und unten waren zu Projektbeginn bereits vorhanden (nicht aber die Services)

Um den Kunden einheitliche Aussagen über die Verfügbarkeit seiner Produkte an deren Standorten zu liefern, sollte ein neues, zentrales System entwickelt werden. Schon mehrmals war der Versuch gescheitert, diese Funktionalität in eines der existierenden Systeme zu integrieren.

Da traf es sich gut, dass zu diesem Zeitpunkt IT-intern ein neuer Technologie-Stack entwickelt wurde, der die IT in die Lage versetzen sollte, die Anforderungen der Fachabteilungen schneller und flexibler zu erfüllen (s. Abb. 1) [Liferay, ZK,FuseESB,GlassFish].

Verbunden mit dem Stack war das Ziel, langfristig eine softwareorientierte Architektur (SOA) in der BSS (Business Support Systems)-Landschaft zu etablieren. In einem klassischen Bottom-up-Ansatz sollte unser Projekt dazu den ersten Schritt machen. Hierzu mussten die technischen Informationen, die für die Verfügbarkeitsabfragen benötigt wurden, von mehreren Services gesammelt, in einer neuen, zentralen Komponente fachlich aufbereitet und dann in Form eines neuen Service diversen Client-Systemen zur Verfügung gestellt werden (s. Abb. 2). Sowohl Backend- als auch Client-Systeme mussten von externen Lieferanten angepasst werden.

Als dritte Neuerung sollte mit diesem Projekt agiles Vorgehen etabliert werden. Konkrete Ziele dahinter waren höhere Flexibilität der IT und damit verbunden ein entsprechend verbessertes Image, eine engere Einbeziehung der Fachbereiche in die IT-Projekte sowie Effizienz- und Qualitätsgewinne. Als Vorgehensmodell wurde Scrum gewählt. Die Rollen des Scrum Masters und des Product Owners wurden vom Kunden selbst besetzt. Im Team arbeiteten im Mittel vier Entwickler sowohl von uns als auch vom Kunden.

Ein neuer Technologie-Stack, ein neues Architektur-Paradigma und ein neues Vorgehensmodell: ziemlich viel für ein Projekt? Auch wenn es auf den ersten Blick so erscheinen mag: Unser Auftrag war nicht, die (IT-)Welt unseres Kunden zu revolutionieren. Unser Auftrag war, einen zuverlässigen Verfügbarkeitsservice zu implementieren. Und vielleicht waren die zusätzlichen Ziele dabei sogar hilfreich.

Agile?

Unser Kunde wollte durch agiles Vorgehen schneller und flexibler auf wechselnde Geschäftsanforderungen reagieren. Den Ausschlag für Scrum als konkrete Methode gab dabei vor allem der hohe Verbreitungs- und Bekanntheitsgrad, womit ausufernde Methodendiskussionen minimiert werden sollten. Von Anfang an war klar, dass Scrum hier als Vehikel zur Einführung von Agilität dienen sollte, ohne selbst Ziel oder Dogma zu sein. Gemäß dem agilen Manifest [Man] gilt ja auch: *„Individuen und Interaktionen über Prozesse und Werkzeuge“*.

Wir starteten mit Scrum nach der reinen Lehre, mit allen Artefakten, Rollen und Meetings. Im Laufe der Zeit mussten wir an einigen Stellen etwas nachjustieren. So merkten wir bald, dass – wie in vielen uns bekannten Scrum-Projekten – unser Product Owner seine Rolle nicht in allen Aspekten ausfüllen konnte. Zwar war er gut verankert in der Organisation und kannte die Fachlichkeit, aber er hatte wenig Erfahrung mit IT-Projekten, keine Scrum-Erfahrung und wenig Zeit. Wir vereinbarten als Gegenmaßnahme, dass ein Teammitglied ihn bei seinen Aufgaben unterstützt. Dies betraf sowohl die Kommunikation mit dem Fachbereich als auch Pflege und Priorisierung des Backlogs. Die fachliche Entscheidungshoheit und die Abnahme der Software im Sprint-Review verblieben dagegen beim Product Owner. Eine riskante Entscheidung, die aber letztlich gut funktionierte.

Eine weitere Schwierigkeit ergab sich aus der Serviceorientierung und der Entscheidung, User-Storys als Backlog-Artefakte zu verwenden. Der Begriff des Users war aus zwei Gründen Auslöser für Diskussionen. Zum einen waren die Hauptbenutzer, wie erwähnt, nicht Menschen, sondern Konsumenten-Systeme der bereitgestellten Services. Zum anderen führten wechselnde Anforderungen auch zu einer grundsätzlich veränderten Sicht auf die Benutzer. Daraus ergaben sich folgende Schwierigkeiten:

- Die User-Storys aus Sicht der Konsumenten spiegelten nur die Anforderungen der Systeme, nicht aber die der tatsächlichen Benutzer wieder und waren somit schwer anwendbar und vorzeigbar. Die Tatsache, dass die Client-Systeme extern und mit teilweise frühzeitig festgelegten Schnittstellen entwickelt wurden, kam hier erschwerend hinzu.
- Technische User-Storys zur Implementierung des Technologie-Stacks waren nur schwer zu formulieren.
- Die Einbindung der Zusammenarbeit mit nicht-agilen Zulieferern ließ sich ebenfalls nur schwer in User-Storys festhalten.

Gegen Ende des Projekts führten vor allem die letzten beiden Punkte dazu, dass wir unseren Scrum-Prozess anpassen mussten. Anstelle von abgeschlossenen Sprints trat eine Kanban-ähnliche Arbeitsweise, durch die vor allem der „Work in Progress“ minimiert wurde. Gerade in der Zusammenarbeit mit dem Betrieb bewährte sich dieses Vorgehen. Der festgelegte Zeitpunkt und Umfang des Go-Lives machte außerdem die genaue fachlich-inhaltliche Planung der Sprints und die Messung der Velocity verzichtbar.

Bleibt die Frage: Haben wir trotz allem noch Scrum gemacht? Die Frage haben wir innerhalb, aber vor allem außerhalb des Teams mehrfach diskutiert bzw. diskutieren müssen. Sie scheint also von einigem Interesse zu sein.

Zum einen bekennen wir uns hierbei klar zu Verfechtern des bekannten „Scrumbut“ [App09]. Zum anderen beharren wir darauf, trotz „but“ durchaus Scrum gemacht zu haben (auch wenn diese Meinung nicht alle teilen). Dies aus folgenden Gründen:

- Trotz aller Stolpersteine sind wir immer agil geblieben.
- Wir kennen wenige Projekte, in denen ein reineres Scrum praktiziert wurde.
- Oder, wie es der Entwicklungsleiter des Kunden ausdrückte: „Wenn ihr schon Scrum-Butt macht, welches Körperteil beschreibt dann unsere anderen Projekte?“

Scrum muss sich als eine agile Methode dem agilen Manifest und damit dem oben erwähnten Prinzip unterwerfen. Individuen und Interaktionen haben bei uns eine Anpassung des Prozesses Scrum erforderlich gemacht.

Das agile Vorgehen hat sich unabhängig davon vollständig bewährt. Wir hatten ständig wechselnde Anforderungen, unerwartete Beeinflussungen durch Nachbarprojekte, eine wechselnde Team-Zusammensetzung sowie unerwartete technische Probleme. Zum Beispiel waren unsere Anwendersysteme am Ende ganz andere als am Anfang geplant. Außerdem wurde die Logik der Verfügbarkeitsermittlung während der Projektlaufzeit grundlegend geändert. Ohne agiles Vorgehen wäre es schwer gewesen wäre, mit diesen Umständen klarzukommen.

SOA?

Wie oben dargestellt, war die Einführung einer serviceorientierten Architektur ein Neben-Projektziel. In einer Vorphase des Projekts wurden die Services grob vordefiniert. Das Auftragsmanagement und die Technik stellten technische Daten bereit, die von einem zentralen Verfügbarkeitsservice aggregiert und an Abnehmer wie das Kundencenter bereitgestellt wurden. Ferner wurde ein zentraler Geodatenservice etabliert, der allen anderen Kernsystemen Geodateninformationen liefert.

Das Design dieser Services erfolgte im Bottom-up-Vorgehen [Han10]. Die Services wurden nur für das Projekt definiert und in der konkreten Entwicklung verfeinert. Während die Zusammensetzung der Services über das gesamte Projekt fast unverändert blieb, änderten sich Semantik und Syntax der einzelnen Services aufgrund der erwähnten wechselnden Anforderungen häufig.

Mehrere vorangegangene Projekte mit ähnlichem Auftrag, aber anderer Architektur waren gescheitert. Diesen Projekten war gemeinsam, dass die ganze Funktionalität jeweils in einem Kernsystem abgebildet wurde. Aus anderen Systemen benötigte Daten wurden per Offline-Schnittstellen repliziert. Der serviceorientierte Ansatz schaffte es, eindeutige Verantwortlichkeiten im Sinne von „Separation of Concerns“ mit Aktualität der Daten zu vereinen. Gleichzeitig waren die Services auch eindeutig den jeweiligen Fachbereichen der Firma zugeordnet.

Serviceorientierung war also für unser Projekt Weg und Ziel gleichzeitig. Das strategische Ziel einer sukzessiven SOA-Einführung wäre streng genommen gar nicht nötig gewesen, weil sich die gleiche Architekturlösung ohnehin ergeben hätte – vorausgesetzt, wir wären auf diese Lösung auch ohne das vorgegebene strategische Ziel verfallen. Darüber zu spekulieren ist müßig, für das Projekt war die SOA-Strategie sicherlich ein Glücksfall.

Dagegen offenbarte der gewählte Technologie-Stack schon früh im Projekt Schwächen. Durch geschickt gewählte Maven Archetypes, Continuous Integration über einen Hudson [Hudson] sowie ein feinmaschiges Netz an Unit- und Servicetests gelang es uns zwar, die Entwicklung vom ESB zu entkoppeln. Es zeigte sich aber, dass der gewählte Applikationsserver (Glassfish 3.0) unter einigen Kinderkrankheiten litt. So verhinderten mehrere Bugs das saubere Zusammenspiel zwischen Enterprise-JavaBeans und CDI (Context Dependency Injection – JSR 299) sowie die Integration von Bean Validation (JSR 303). Die Anbindung von ZK wurde statt CDI mit einem einfachen Service-Locator-Muster gewährleistet.

Gegen Ende des Projekts wurde außerdem deutlich, dass der gewählte ESB den Stabilitätsanforderungen nur bedingt genügte. Vor allem das Deployment bereitete erhebliche Kopf-

schmerzen. Zusätzlich erwies sich das Zusammenspiel der beteiligten Komponenten (Apache Servicemix, Apache Camel, CXF) als fehleranfällig. Es kostete z. B. unangemessen viel Zeit, den Camel-Routen eine angemessene Anzahl Threads zuzuweisen, um zu verhindern, dass durch das Beschreiten mehrerer Routen Livelocks innerhalb des ESBs entstehen konnten.

Der vorgegebene Technologie-Stack hat sich damit als Schwachpunkt in der Strategie erwiesen. Wir haben ihn während des Projekts modifiziert und vereinfacht. Auch nach Go-Live plant der Kunde noch weitere Maßnahmen zur Verschlankung. Beeinträchtigt das die weitere SOA-Einführung? Wir glauben nicht. Wie auch Thomas Erl in seinem Buch [Erl08] schreibt, sollte die Eignung einer Umgebung für die Problemlösung im Vordergrund stehen, auch bei einer SOA. Unsere derzeitigen Anforderungen konnten wir mit dem angepassten Stack umsetzen. Bei neuen Anforderungen werden die Karten neu gemischt.

Die Regeln des agilen Software-Designs gelten auch für eine SOA. Wichtiger als die Technologie ist die Angemessenheit der fachlichen Services. Für uns führt dies zu der Schlussfolgerung, dass der Gedanke „Serviceorientierung" deutlich wichtiger ist als die konkrete Umsetzung. Das Ziel der Einführung einer SOA wurde erreicht, aber im Sinne einer anforderungsorientierten, flexiblen Architektur

Agile oder SOA?

Zum Abschluss wollen wir noch die Wechselwirkung von agilem Vorgehen und SOA, wie wir sie im Projekt erlebt haben, diskutieren. Ähnliche Diskussionen finden sich auch in [Bei10] und [Els07].

Agile Welt und SOA-Umfeld haben unterschiedliche Auffassungen von Architektur. Wie viele andere agile Methoden vermeidet auch Scrum ein BDUF (Big design up front) und stellt die Veränderung in den Vordergrund. Große strategische Architekturen widersprechen diesem Anspruch. Dagegen ist die Einführung einer SOA normalerweise mit einer entsprechenden strategischen Top-down-Betrachtung verbunden und deutlich architekturgetrieben. Auch wenn die konkrete Implementierung durchaus bottom-up getrieben sein kann, steht hier doch die Architektur immer vor dem Konzentrieren auf mögliche Veränderungen.

In unserem Projekt erwies es sich aber nicht nur als möglich, beide Betrachtungen der Architektur zu vereinen. Darüber hinaus ergänzten sich die jeweiligen Vorteile sogar. Wie erwähnt, brachten die gewählten Technologien große Probleme mit sich. Durch die saubere Definition der fachlichen Services gelang es uns aber immer, Änderungen in der Technologie effizient durchzuführen. Die Zusammensetzung der Services und die Verantwortlichkeiten sind unabhängig von der Technik und erlauben für die Zukunft weitere Veränderungen, wie z. B. die Umstellung von SOAP auf REST. Zusammengefasst war die Flexibilität, die die Serviceorientierung mit sich brachte, ein essenzieller Faktor, der uns ermöglichte, so agil auf technische Schwierigkeiten zu reagieren.

Auch umgekehrt stellten wir Synergieeffekte fest: Zwar blieb die Grundstruktur der Services über das Projekt fast unverändert, jedoch nicht die Syntax und die Semantik der einzelnen Methoden. Wechselnde Projektziele, ein sich vergrößerndes Verständnis der Fachlichkeit sowie Diskussionen zwischen den einzelnen Fachbereichen führten zu oft geänderten Definitionen der Services. Nach unserem Verständnis hätten fest definierte Service Contracts zu Beginn des Projekts es unmöglich gemacht, auf diese Änderungen zu reagieren. Die Agilität führte dazu, dass auch die Schnittstellen auf diese Änderungen reagieren konnten.

Die Grenzen dieser Agilität wurden jedoch in der Kooperation mit einigen der weniger agilen Zulieferern deutlich. Hier führten bereits kleine Änderungen zu fortschreitend heftigeren Termin- und Budgetdiskussionen – auch, weil keine ausreichenden automatisierten Regressionstests zur Absicherung der Änderungen vorhanden waren, wie sie in agilen Projekten Standard sind.

Zusammenfassung

Den größten Beitrag zum Erfolg in unserem Projekt hatte unserer Meinung nach die Agilität, also der Wille und die Möglichkeit, alle Parameter im Projekt zu variieren, um den Projekterfolg zu gewährleisten. Dem fielen der vorgegebene Technologie-Stack und Teile des Vorgehensmodells zum Opfer. Interessanterweise war eine der stabilsten Größen die fachliche Service-Architektur. Sie ermöglichte und unterstützte Agilität auf allen möglichen anderen Ebenen. Letztlich ist das keine überraschende Aussage, denn ein Ziel von SOA ist es, fachliche Flexibilität zu ermöglichen. Diese Flexibilität hat uns bereits zur Entwicklungszeit geholfen.

Literatur und Links

[App09] J. Appelo, ScrumButs are the Best Part of Scrum. `www.noop.nl/2009/09/scrumbuts-are-the-best-part-of-scrum.html`
[Bei10] M. Beijleveld, Agile and SOA, Hand in Glove?, InfoQ, 2.2.2010, `www.infoq.com/articles/agile-soa`
[Els07] A. Elssamadisy, SOA and Agile: Friend or Foes?, InfoQ, 14.4.2007, `www.infoq.com/articles/SOA-Agile-Friends-Or-Foes`
[EnLe11] M. Engler, M. Lerch, Scrum einführen, in: JavaSPEKTRUM, 4/2011
[Erl08] Th. Erl, SOA: Entwurfsprinzipien für serviceorientierte Architektur, Addison-Wesley Verlag, 2008
[FuseESB] `www.fusesource.com/products/enterprise-servicemix`
[GlassFish] `www.glassfish.java.net`
[Han10] I. Hanschke, Symbiose zwischen EAM und SOA, in: SOA Workshop Leipzig - Tagungsunterlagen, 2010, s. a. `http://www.iteraplan.de/sites/all/files/docs/Hanschke_SymbioseEAMSOA_2010.pdf`
[Hudson] Hudson Contiuous Integration Server, `www.hudson-ci.org`
[Liferay] Liferay Portal, `www.liferay.com`
[Man] Agiles Manifest, `www.agilemanifesto.org`
[ZK] ZK Framework, `www.zkoss.org`

Dominik Rose ist Senior Softwareentwickler bei der iteratec GmbH. Nach seinem Abschluss an der RWTH Aachen hat er als Entwickler die großen Unterschiede zwischen Theorie und Praxis kennengelernt. Er beschäftigt sich mit agilen Vorgehensweisen wie Scrum und Kanban, um effektive und flexible Antworten auf die praktischen Herausforderungen in Entwicklungsprojekten zu finden.
E-Mail: Dominik.Rose@iteratec.de

Dr. Karl-Heinz Wichert ist Chef-Architekt bei der iteratec GmbH. Er hat als Entwickler, Architekt und Projektleiter in vielen Softwareentwicklungsprojekten die Vorzüge agiler Entwicklung schätzen gelernt. Neben der operativen Softwareentwicklung beschäftigt er sich mit Softwarequalität und der Frage, wie man sie kontrollieren kann, obwohl sie nicht messbar ist.
E-Mail: Karl-Heinz.Wichert@iteratec.de