Volker Hoyer, Christoph Schroth, Katarina Stanoevska-Slabeva, Till Janner

Web 2.0-Entwicklung – ewige Beta-Version

Eine neue Generation von internetbasierten Community-Plattformen wie YouTube, Flickr oder del.icio.us hat in den letzten Jahren großes Interesse in Forschung und Industrie hervorgerufen. Diese Plattformen beziehen den Nutzer als zentralen Teil des Applikationsdesigns explizit mit ein und prägen einen neuen Trend in der Entwicklung von Web-Anwendungen. Zahlreiche Funktionalitäten, die exakt den Bedürfnissen der Nutzer entsprechen, können so bereitgestellt und kontinuierlich weiterentwickelt werden. Dem Paradigma der ewigen Beta-Version entsprechend, stellen Web 2.0-basierte Anwendungen keine statischen Softwareartefakte mehr da. Sie sind vielmehr dezentrale Dienste, die sowohl von Nutzern als auch von Betreibern ständig angepasst werden können. Die fortwährende Anpassung von Web 2.0-Plattformen stellt insbesondere an den Entwicklungsprozess neue Anforderungen. Durch Anwendung von klassischen Methoden der Softwareentwicklung können diese nicht erfüllt werden. Dieser Artikel beschreibt damit verbundene Herausforderungen und mögliche Lösungsansätze und illustriert diese mit zahlreichen Beispielen.

Inhaltsübersicht

1 Motivation

Eine neue Form von Community-Plattformen hat in den letzten Jahren großes Interesse hervorgerufen. Dieses Phänomen wird unter dem Begriff Web 2.0 subsumiert. Angebote, die rein auf benutzergeneriertem Content beruhen, wie z. B. YouTube, Flickr, del.icio.us, sowie Plattformen zur Umsetzung von Mash-ups stellen nur einige Beispiele für diese Anwendungen dar.

Tim O'Reilly gilt als derjenige, der maßgeblich den Begriff Web 2.0 geprägt hat. Gemäß [O'Reilly 2005] lässt sich die Web 2.0-Philosophie durch sieben wesentliche Eigenschaften beschreiben: Das Web wird zunächst als umfassende Plattform für die Entwicklung von eng miteinander verbundenen Systemen betrachtet (1), die über gemeinsame Protokolle realisiert werden, offene Standards benutzen und miteinander gemäß definierten Vereinbarungen kooperieren. Die Nutzbarmachung kollektiver Intelligenz der Web-Nutzer (2), die Generierung und Verwaltung von anwendungsrelevanten, nutzerspezifischen Daten (3) sowie das Ende eines festen Software-Releasezyklus (4) werden als weitere Charakteristika angeführt. Die Verwendung von agilen Programmiermethoden (5), der Einsatz verschiedener Medien und Endgeräte für die Nutzung dieser internetbasierten Anwendungen (6) und die Realisierung einer »Rich User Experience« (7) stellen weitere zentrale Konzepte der Web 2.0-Philosophie dar. Im Rahmen einer detaillierten Untersuchung von 40 Web 2.0-Anwendungen von [Högg et al. 2006] werden die vorgestellten Charakteristika zu folgender Begriffssynthese zusammengefasst, die als grundlegende Definition im Rahmen dieses Artikels verstanden werden soll: »*Web 2.0 ist definiert als die Philosophie der gemeinschaftlichen Maximierung kollektiver Intel-*

ligenz und der Schaffung eines Mehrwertes für jeden Teilnehmer durch eine formalisierte und dynamische Erzeugung und gemeinsame Nutzung von Informationen« [Högg et al. 2006]. Grundlage dafür sind Web 2.0-Plattformen, die eine einfache Erstellung, Kombination, Verwaltung, Bewertung und Nutzung von benutzergenerierten Inhalten erlauben.

Im Gegensatz zu konventioneller Software unterliegen Web 2.0-Plattformen keinen statischen und langen Releasezyklen. Diese entwickeln sich vielmehr kontinuierlich weiter und können auch vom Nutzer selbst signifikant mit beeinflusst werden. Versionen, lokale Installationen oder Updates spielen hier keine Rolle. Die eigentliche Servicefunktionalität tritt in den Vordergrund und nicht mehr die zu installierende Applikation [Musser & O'Reilly 2006]. Dieses neue Paradigma der Bereitstellung von Softwareservices prägt zunehmend auch die Softwareentwicklung in Unternehmen [Zarnekow et al. 2006]. Eine erfolgreiche Umsetzung dieses Konzepts ist allerdings abhängig von der Berücksichtigung der veränderten Rahmenbedingungen gerade bei der Entwicklung von dynamischen, sich ständig verändernden Plattformen und Services.

Ziel dieser Arbeit ist es, die sich bei der Web 2.0-Serviceentwicklung ergebenden Herausforderungen in verschiedenen Dimensionen des Entwicklungsprozesses zu identifizieren und zu analysieren sowie neuartige Lösungsansätze aufzuzeigen. Kapitel 2 analysiert dazu die Veränderungen in der Web 2.0-Serviceentwicklung bezüglich der Dimensionen Vorgehensmodell, Softwarearchitektur und Management. In Kapitel 3 wird ein umfassendes Modell für Serviceentwicklung gemäß dem Web 2.0-Paradigma vorgestellt, das auf den zuvor erarbeiteten Prinzipien aufbaut. Kapitel 4 schließt den Artikel mit einer Zusammenfassung und einem kurzen Ausblick auf zukünftige Herausforderungen ab.

2 Herausforderungen der Web 2.0-Serviceentwicklung

Im Allgemeinen umfasst der Begriff Softwareentwicklung »*... alle Entwicklungs- und Managementaktivitäten, die ausgeführt werden müssen, um ein Softwareprodukt und damit verbundene Serviceleistungen zu erzeugen*« [Trittmann et al. 1999]. Die Organisation der Softwareentwicklung wird durch drei Faktoren beeinflusst:

- das Vorgehensmodell,
- das Entwicklungsparadigma und die daraus resultierende Softwarearchitektur und
- der spezifische Managementansatz für den Softwareentwicklungsprozess.

Ein Vorgehensmodell (synonym auch Phasenmodell, Entwicklungsmodell oder Software Life Cycle genannt) beschreibt in idealisierter und generalisierter Form einen Softwareentwicklungsprozess oder den Softwarelebenszyklus. Es definiert die einzelnen Phasen der Softwareentwicklung, die Reihenfolge der Aktivitäten, deren Ziele, Inputs und Outputs und schließlich deren Zusammenhang. Zur Entwicklung von konkreten Softwareartefakten wird das allgemeine Vorgehensmodell innerhalb eines konkreten Projektmanagements umgesetzt und durch weitere Managementaspekte ergänzt. Innerhalb des Projektmanagements werden konkrete Zeitpläne, Ressourcen, Mitarbeiter und wirtschaftliche Ziele definiert (siehe dazu [Jurison 1999]).

Die aktuelle Entwicklung hin zu dynamisch veränderbaren Web 2.0-Plattformen und den darüber angebotenen Services stellt den Prozess der Softwareentwicklung und dessen Management vor neue Herausforderungen [Lowe & Henderson-Sellers 2001; Zarnekow et al. 2006]. In den nachfolgenden Abschnitten erfolgt eine detaillierte Analyse dieser Herausforderungen. Außerdem werden erste Lösungsansätze anhand der oben identifizierten Faktoren vorgestellt.

2.1 Vorgehensmodell

Traditionelle Vorgehensmodelle aus der Softwareentwicklung wie beispielsweise das Wasserfallmodell, das Spiralmodell, das V-Modell oder der Rational Unified Process (RUP) verfolgen schwerfällige Prozesse und starre Vorgehensweisen entlang zuvor klar definierter Pläne. Diese konventionellen Modelle haben sich für die Entwicklung klassischer Anwendungen bewährt, unterstützen den angedeuteten, hoch dynamischen Produktentwicklungsprozess von Web 2.0-Anwendungen jedoch nur unzureichend. Agile Methoden dagegen verfolgen das Ziel einer »people-centric« Softwareentwicklung. Sie stellen sowohl den formalen Prozess als auch die bei der Entwicklung angewandte Technologie in den Hintergrund zugunsten der beteiligten Personen und deren Interaktion. Alle agilen Vorgehensmodelle wie z. B. eXtreme Programming (XP), Scrum, Crystal (family of methodologies), Feature Driven Development (FDD), Adaptive Software Development (ASD) oder Open Source Software Development (OSSD) sind nach [Abrahamson et al. 2002] durch identische Eigenschaften charakterisiert (siehe auch [Highsmith 2002]), die aus dem Agile Manifesto (*http://agilemanifesto.org*) abgeleitet sind. Unterschiede zwischen den Methoden bestehen nur in der verfolgten Priorisierung der Charakteristika. Die gemeinsamen Charakteristika können wie folgt zusammengefasst werden:

Inkrementelle Erweiterungen. Web 2.0-Applikationen erfordern inkrementelle Erweiterungen der bestehenden Funktionalität in besonders kurzen Software-Releasezyklen. Klassische Methoden der Softwareentwicklung sehen eine detaillierte Spezifikation des Gesamtsystems vor dem Beginn der Implementierungsphase vor. Im Gegensatz dazu basieren agile Methoden der Softwareentwicklung auf Umsetzung kleiner Entwicklungsaufgaben innerhalb von kurzen Zeiträumen ohne eine vorgängige vollständige Ausarbeitung des finalen Produktes. Je nach Entwicklungspfad werden einzelne Komponenten später erweitert bzw. modifiziert. Zwei Fallbeispiele verdeutlichen die Bedeutung dieses Prinzips. Ende 2006 hat der Betreiber der Plattform für soziales Networking, »XING« (*www.xing.com*, früher OpenBC), seinen Dienst grundlegend verändert und seine Nutzer mit einer neuen Benutzeroberfläche und neuen Funktionalitäten konfrontiert. Anstatt inkrementell neue Servicekomponenten einzuführen und ihre Attraktivität und Akzeptanz direkt von Nutzern evaluieren zu lassen, wie das von erfolgreichen Onlineplattformen wie eBay oder Google praktiziert wird, hat eine traditionelle radikale Umstellung der gesamten Plattform zu teils kritischen Reaktionen in der Community geführt. Die Markteinführung des »Google Maps«-Service erfolgte dagegen stufenweise, sodass das Feedback der Nutzer kontinuierlich in die Gestaltung des neuen Produktes integriert werden konnte. Der Service konnte so dauerhaft auf Kundenbedürfnisse ausgerichtet werden und erfreut sich ausgesprochen großer Beliebtheit.

Verzicht auf umfangreiche formale Dokumentationen. Im Gegensatz zu klassischen Methoden, die auf zahlreichen fest definierten Dokumenten (beispielsweise Anforderungsdefinitionen, Spezifizierungen, Beschreibungen von Anwendungsszenarien, Glossare, Entwurfs- oder Interaktionsdiagramme) aufbauen, soll hier die Konzentration auf eine lauffähige Software (ggf. Prototypen) gerichtet werden, die ein früheres Feedback der Kunden ermöglicht.

Enge Zusammenarbeit mit dem Auftraggeber. Die von zahlreichen agilen Methoden proklamierte enge Zusammenarbeit zwischen Auftraggeber und Auftragnehmer einer Applikation ist ein weiterer wichtiger bestimmender Faktor für Vorgehensmodelle im Bereich der Web 2.0-Serviceentwicklung. Durch eine konstante und enge Zusammenarbeit zwischen den Nutzern und den Anbietern von Web 2.0-Plattformen wird der Nutzer explizit zum Bestandteil des Entwicklungsprozesses und kann das eigentliche Produkt so maßgeblich aktiv mitgestalten [Highsmith 2002].

Ständige Anpassungen. Die regelmäßige Diskussion möglicher Anpassungen der Plattform während des Produktlebenszyklus sind der inflexiblen Ausrichtung nach einem vordefinierten Plan vorzuziehen. Der eigentliche Mehrwert von Web 2.0-Applikationen besteht in der dauerhaften Weiterentwicklung und in der Anpassung an aktuelle Nutzeranforderungen. An einem einmal festgelegten Plan festzuhalten, wäre dem Erfolg eines solchen Produktes nicht zuträglich. Dies gilt insbesondere für Web 2.0-Plattformen, die neue Funktionalitäten anbieten und auf unbekannte globale Zielgruppen ausgerichtet sind. Es ist fast unmöglich, für solche Plattformen Anforderungen adäquat im Voraus zu analysieren und zu definieren [Lowe & Henderson-Sellers 2001].

Operative Phase. Zusätzlich zu den oben vorgestellten Eigenschaften der agilen Methoden spielt die sogenannte operative Phase eine wesentliche Rolle in der Entwicklung von Web 2.0-Plattformen. Ein immer verfügbarer Onlineservice verlangt, Entwicklungsaktivitäten auf die Phase des operativen Einsatzes auszudehnen [Musser & O'Reilly 2006]. Web 2.0-Anwendungen stellen keine statischen Artefakte dar, die nach einer klar definierten Entwicklungsphase nicht mehr modifiziert werden, sondern werden potenziellen Nutzern vielmehr als flexible und über das Web verfügbare Services angeboten.

[Hinchcliffe 2006] propagiert die Anwendung von Prinzipien des »Lean Management« bzw. der »Lean Production« auf die Serviceentwicklung im Bereich Web 2.0, wie in Abbildung 1 dargestellt. Die beteiligten Individuen – Entwickler sowie Nutzer – auf der einen Seite und der Service bzw. das Produkt auf der anderen Seite nehmen dabei gegenseitig Einfluss auf den Entwicklungsprozess und müssen eng aufeinander abgestimmt werden. Eine verringerte Komplexität verbunden mit einem schnelleren Feedback erzeugt einen neuen Wettbewerbsvorteil gegenüber konventionellen Ansätzen der Softwareentwicklung.

Um ein Produkt in einem verkürzten Zeitraum zu entwickeln, sollen über eine verringerte Abstraktionsebene und mit einer verschlankten Entwicklung sowohl personelle als auch finanzielle Ressourcen eingespart werden. Wie Abbildung 1 zu entnehmen ist, spielt nicht nur die enge Verbindung zwischen Nutzern und Entwicklern, sondern auch die Einhaltung von bestimmten Prinzipien bei der Spezifikation der

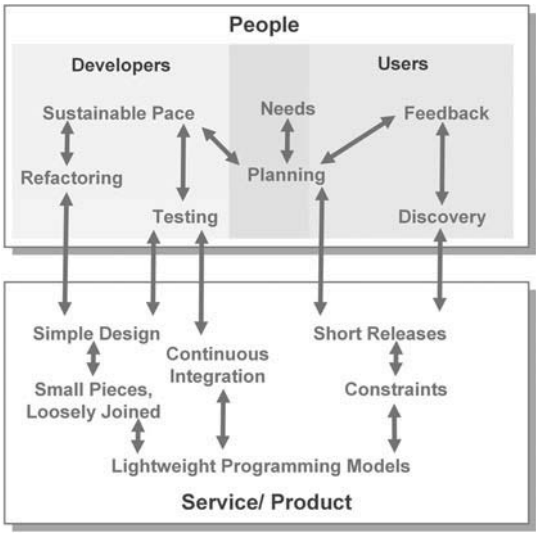

Abb. 1: Agile/ Lean Development [Hinchcliffe 2006]

Architektur von Web 2.0-Applikationen selbst eine entscheidende Rolle. Die oben angedeuteten plattformunabhängigen Protokolle, die modulare Bauweise lose gekoppelter Applikationsbausteine sowie die nahtlose Integration von Backend-Funktionalität, die vor dem eigentlichen Nutzer verborgen bleibt, stellen drei wichtige Architekturprinzipien dar und werden im Folgenden detailliert diskutiert.

2.2 Entwicklungsparadigma und Softwarearchitektur

Wie bereits im vorherigen Abschnitt angedeutet und auch in Abbildung 1 dargestellt, implizieren die neue Vorgehensweise und die veränderten Rahmenbedingungen neue Entwicklungsparadigmen und neue Anforderungen an die Softwarearchitektur.

Plattformunabhängige, dynamische Sprachen wie Python, PHP oder Ruby sowie einfache Schnittstellenbeschreibungen und Protokolle wie Really Simple Syndication (RSS) oder Representational State Transfer (REST) haben sich im Umfeld von Web 2.0 etabliert. Sie vereinfachen die Entwicklung und die Integration, aber auch das Testen sowie die Wiederverwendung von Applikationsbausteinen und reduzieren so Entwicklungskosten aufgrund erhöhter Effizienz [Hinchcliffe 2006]. Eine charakteristische Technologie zur Verbesserung der Benutzbarkeit von Benutzeroberflächen in der Web 2.0-Entwicklung ist AJAX (Asynchronous JavaScript and XML). Diese ermöglicht die Übertragung von Designkonzepten von Desktop-Anwendungen auf webbasierte Anwendungen und führt zu einer neuen »Rich User Experience« wie von [O'Reilly 2005] propagiert.

Kleine, lose gekoppelte Services. Die Bereitstellung einfach aufgebauter und leicht verwendbarer Application Programming Interfaces (APIs) stellt einen einfachen Zugriff auf im Web verteilte Ressourcen sicher. Feingranulare und untereinander lose gekoppelte Services führen so zu einer hohen Flexibilität des Produktes in Bezug auf Änderungen, die aufgrund sich wandelnder Nutzerbedürfnisse vorgenommen werden müssen [Musser & O'Reilly 2006]. Der daraus resultierende hohe Grad an Modularität und Flexibilität ermöglicht es, Services zu modifizieren, ohne die Gesamtanwendung zu beeinflussen und ohne das Vertrauen der Nutzer in die Zuverlässigkeit eines Service zu erschüttern.

Integration von Backend-Anwendungen. Die Verwendung von plattformunabhängigen Protokollen und feingranularen, lose gekoppelten Services beeinflusst direkt die Wahrnehmung des Service durch den Nutzer. Klassische Applikationskomponenten im Backend existieren im Web 2.0-Umfeld weiterhin, bleiben jedoch vor den Usern verborgen. Lowe und Henderson-Sellers verwenden für diese Vorgehensweise den Begriff »Landscape gardening«: »*Web site development is often about creating an infrastructure (laying out the garden) and*

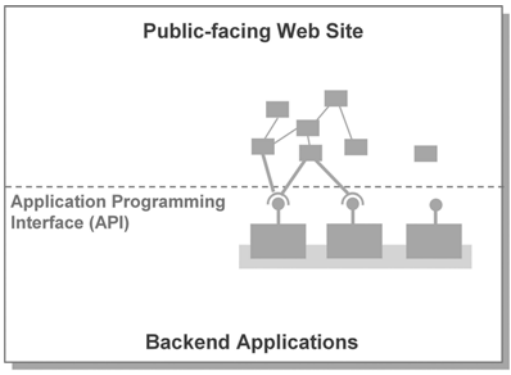

Abb. 2: Softwarearchitektur

then ›tending‹ the information that grows and blooms within this garden. Over time the garden will continue to evolve, change and grow. A good initial architecture should allow this growth to occur in a controlled and consistent manner« [Lowe & Henderson-Sellers 2001]. So ist Internetnutzern die einfache Oberfläche des Google-Service gut bekannt, die darunter verborgene Komplexität der sogenannten nicht funktionalen Anwendungskomponenten bleibt jedoch unsichtbar. Sicherheit und Skalierbarkeit der Services werden zu diesen Komponenten gezählt, die langsameren Veränderungsprozessen unterliegen und von Web 2.0-Applikationsbetreibern im Hintergrund unter Verwendung eher klassischer Vorgehensmodelle entwickelt werden. Hier werden auch klassische Vorgehensmodelle eingesetzt, weil weniger Agilität und Simplizität als vielmehr Performanz und Zuverlässigkeit entscheidend sind. Wichtig ist die Trennung von Anwendungen im Backend und von solchen, die für den Nutzer direkt sichtbar sind und schnellen Veränderungen unterliegen können. Dies kann über klar definierte Schnittstellen realisiert werden, die die Verwendung verschiedener Architekturkonzepte ermöglichen.

2.3 Management

Dynamische, sich kontinuierlich an Nutzerbedürfnisse anpassende Softwareservices erfordern neuartige Managementansätze. Konventionell erfolgt die Softwareentwicklung innerhalb der Grenzen von klar definierten und abgegrenzten Projekten. Die Vorgehensmodelle werden dabei durch spezifisches Projektmanagement der Softwareentwicklung [Jurison 1999] operationalisiert. Projekte haben gewöhnlich ein definiertes Ende, und es besteht eine klare Trennung zwischen Projektmanagementaufgaben während der Entwicklung und der Wartung nach der Inbetriebnahme der Software. Die Entwicklung hochdynamischer Web 2.0-Plattformen dagegen weist kein klares Ende auf. Wie bereits oben beschrieben, verschmelzen die Phasen der Konzipierung, Entwicklung und Nutzung in einem einzigen kontinuierlichen und agilen Prozess. Diese Art von Softwareentwicklung erfordert einen entsprechend ganzheitlichen Managementansatz, der eher einem kontinuierlichen Produktmanagement als einem konventionellen Projektmanagement gleicht [Zarnekow et al. 2006]. Das Produktmanagement von Web 2.0-Plattformen ist nicht auf ein Projektende fixiert, sondern auf Kundenwünsche, ohne dabei wirtschaftliche Aspekte zu vernachlässigen. Neue Anforderungen sind in dem Produkt kontinuierlich mit aufzunehmen, um zum einen die Produktqualität zu verbessern und zum anderen zusätzlichen Nutzen als Differenzierungsmerkmal zu generieren.

Die dritte und letzte Dimension unserer Analyse betrachtet das Management der Web 2.0-Services. Sowohl klassische als auch agile Methoden schenken dieser Teildisziplin der Softwareentwicklung nur eingeschränkte Aufmerksamkeit. Gerade in der Web 2.0-Ära nimmt diese jedoch aufgrund der Ausdehnung der Entwicklung auf die operative Phase und wegen der inkrementellen Entwicklung in kurzen Releasezyklen eine entscheidende Rolle ein. Die wichtigsten Aspekte des kontinuierlichen Produktmanagements sind: Verwischen von Grenzen und Neuverteilung der Rollen in dem Entwicklungsprozess, Management der Benutzer-Community, Sicherung der Qualität, Management des Risikos und Entwicklung einer ganzheitlichen Produktpolitik. Diese Aspekte werden nachfolgend kurz beschrieben.

Verwischen von Grenzen. Das Aufkommen von Web 2.0-Anwendungen hat die Rolle der Nutzer bei der Serviceentwicklung grundlegend verändert. Diese werden in die Gestaltung und Weiterentwicklung explizit mit einbezogen und übernehmen eine neue Rolle als Co-Entwickler und Echtzeittester [Musser & O'Reilly 2006]. Anregungen und Echtzeit-Feedback ermöglichen es, die Services kontinuierlich und bei gleichzeitig verringerter Reaktionszeit weiterzuentwickeln und direkt an den Kundenwün-

schen auszurichten [Hinchcliffe 2006]. Eine vergleichbare Rollenveränderung ist auch bei den traditionellen Entwicklern zu beobachten. Anstatt die Verantwortung für ein Softwareartefakt nach dessen Fertigstellung an andere Unternehmenseinheiten abzugeben, müssen Entwickler im Web 2.0-Bereich ein deutlich breiteres Tätigkeitsspektrum abdecken. Konzeption, Realisierung, aber auch Betrieb, Wartung und die Berücksichtigung von Nutzer-Feedback können nun nicht mehr klar voneinander getrennt werden und werden oftmals von den gleichen Personen wahrgenommen.

Community. Wie in der Betriebswirtschaft mit dem Wandel von einem Verkäufer- zu einem Käufermarkt beschrieben, nehmen die Kunden der Services – die Internetnutzer – eine neue Rolle bei der inkrementellen Entwicklung ein. Durch eine gezielte Moderation der öffentlichen Diskussionsrunden ist ein neuer direkter Kommunikationskanal zu den Kunden aufzubauen. Ziel ist es, dass sich die Nutzer mit den Services identifizieren und sich eine Community um die Services bildet. Besondere Bedeutung kommt dabei aktiven Nutzern zu, die sich direkt an der Entwicklung oder durch direktes Feedback an der Serviceverbesserung beteiligen. Neben traditionellen finanziellen Anreizsystemen ist die Nutzerbindung durch Servicezusatznutzen zu ergänzen, die ausgewählten Benutzergruppen als Gegenleistung für ihre Mitarbeit zur Verfügung gestellt werden. Eine Herausforderung in Zusammenhang mit der Community besteht in der möglicherweise großen Teilnehmeranzahl. Die Nutzer, die als freie Entwickler agieren und die verteilte Web 2.0-Serviceentwicklung aktiv begleiten, müssen adäquat koordiniert werden. Dabei sind die gesammelten Erfahrungen aus der Open-Source-Bewegung als Grundlage heranzuziehen.

Qualitätssicherung. Wichtig im Zusammenhang mit der veränderten Rolle der Nutzer als Co-Entwickler und Echtzeittester ist das Bewusstsein, dass die Nutzer die Qualitätssicherung nicht ersetzen, sondern nur die Funktionalität der Services validieren und präzisieren können. Der Betreiber der Services muss den dauerhaften Betrieb in einer Weise sicherstellen, die den Qualitätserwartungen der Nutzer entspricht. So verlangen die kontinuierlichen Updates von Web 2.0-Services (Beta-Version) nach einem disziplinierten Build-, Deployment- und Supportprozess. Eine fehlende Servicequalität birgt die Gefahr, wertvolles Kundenvertrauen zu zerstören und verbunden mit der Offenheit des Internets diese an Wettbewerber umgehend zu verlieren. Als Beispiel kann der »Software as a Service (SaaS)«-Anbieter Salesforce (*www.salesforce.com*) aufgeführt werden, der Anfang 2006 vermehrt Probleme mit der Verfügbarkeit seines Service hatte. Wie von O'Reilly beschrieben, müssen deshalb bereits im Rahmen des Serviceentwicklungsprozesses Technologien berücksichtigt werden, die zum Beispiel durch gleichmäßige Lastaufteilung auf verteilte Ressourcen die Skalierbarkeit eines Dienstes garantieren und so dauerhafte Verfügbarkeit sichern [Musser & O'Reilly 2006].

Risikomanagement. Die Qualitätssicherung muss durch ein aktives Risikomanagement ergänzt werden. Die Erweiterung der Softwareentwicklungsphase um eine operative Phase macht eine Etablierung von Mechanismen zur Identifizierung von Risiken (wie zum Beispiel unzureichende Verfügbarkeit) auch nach der Fertigstellung der ersten Version eines Service notwendig. Ein Frühwarnsystem ist in der operativen Ebene einzurichten und in der Web 2.0-Serviceentwicklung als integraler Bestandteil zu verstehen.

Produktpolitik. Neben diesen klassischen Managementaspekten ist zudem die Auswahl und Weiterentwicklung eines Web 2.0-Service sowie dessen Vermarktung ein wichtiger Teilaspekt des Managements der Softwareentwicklung in der Web 2.0-Ära. So sind entsprechend der Definition der Produktpolitik im engeren Sinne verschiedene Strategien bei der Serviceentwicklung zu verfolgen: Produktinnovation (Entwicklung neuer Services), Produktvarianten

(Verbesserung bestehender Services), Produktdifferenzierung (Ergänzung eines Service), Produkteliminierung (Service wird eingestellt) und Produktdiversifikation (Aufnahme neuer Services mit horizontaler, vertikaler oder lateraler Beziehung zum existierenden Angebot). Anders als auf klassischen Softwaremärkten hat man es bei Web 2.0 mit dem »Long Tail« der Internetnutzer zu tun. Diese lassen sich nur schwer bezüglich Marktgröße, Präferenzen oder Anforderungen einschätzen. Die hohe Individualität ist durch gezielte Protokollierungen und Befragungen über die Webseite zu analysieren, um einen passenden, individuellen Service für die heterogenen Nutzergruppen anbieten zu können.

Planung und Budgetierung. Im Gegensatz zu klassischer Softwareentwicklung, die den Entwicklungsprozess als ein abgeschlossenes Projekt versteht, ist die Web 2.0-Serviceentwicklung, wie zuvor bereits angedeutet, durch eine Produktentwicklung gekennzeichnet. Ein fehlender klar definierter Zeitrahmen und die sich ständig ändernden Kundenwünsche implizieren neue Anforderungen hinsichtlich der Planung und Budgetierung eines Web 2.0-Service. Abhängig von den vorhandenen Ressourcen (Budget und Entwicklungskapazitäten) ist die Produkt- bzw. Serviceentwicklung zu planen und zu priorisieren.

3 Serviceentwicklungsmodell in der Web 2.0-Ära

Auf Grundlage der analysierten und identifizierten Herausforderungen von Web 2.0-Anwendungen im vorherigen Kapitel stellt Kapitel 3 ein Serviceentwicklungsmodell als ersten möglichen Lösungsansatz vor. Eine erfolgreiche Web 2.0-Serviceentwicklung ist, wie bei der Analyse aus dem zweiten Kapitel herausgearbeitet, von der Unterstützung eines ewigen Beta-Entwicklungskreislaufs abhängig, um schneller Services in die Community der Nutzer zu tragen. Ziel ist es, eine enge Beziehung zu den Kunden aufzubauen, Risiken zu reduzieren, die Reaktionsfähigkeit bzw. -sensibilität zu verbessern und auf der Basis von Echtzeitdaten quantifizierbare Entscheidungen zu fällen [Musser & O'Reilly 2006].

Eine entscheidende Rolle bei der Entwicklung nimmt dabei die neu identifizierte Rolle der Nutzer als Co-Entwickler und Echtzeittester ein. Zur operativen Zeit werden Nutzern dabei inkrementell neue und erweiterte öffentliche Services als Beta-Version angeboten. Risiken hinsichtlich der Qualität neuer bzw. angepasster Services sollen dadurch minimiert werden, dass nur ausgewählte Nutzergruppen diese Services in den ersten Entwicklungsstadien als Pilotapplikationen nutzen dürfen. Sogenannte »Core User« können insofern als freie Entwickler charakterisiert werden, als sie aktiv an der Entwicklung neuer Services teilnehmen. Bei »Premium User« handelt es sich um eine Benutzergruppe, die besonders aktiv an öffentlich zugänglichen Diskussionen teilnimmt und versucht, durch ihr unmittelbares Feedback die angebotenen Services bzw. Produkte zu verbessern. Über eine moderierte Diskussionsplattform (beispielsweise Bug-Tracking-Systeme wie Bugzilla (*www.bugzilla.org*) oder Blogs) fließt Feedback der jeweiligen Nutzer direkt in die Formulierung der Anforderungen an die Entwicklung ein (siehe Abb. 3). Nach Selektierung und Priorisierung der funktionalen Nutzeranforderungen und unter Berücksichtigung der Planungs- und Budgetaspekte sowie der verfolgten Produktpolitik durch das Management sind die Anforderungen umzusetzen und anschließend über einen disziplinierten Build- und Deploymentprozess den Nutzern anzubieten.

Wie oben bei der Analyse der Softwarearchitektur aufgezeigt, greifen die kleinen, lose gekoppelten Web 2.0-Services auf komplexe Backend-Applikationskomponenten über standardisierte Schnittstellen (APIs) zu. Diese Backend-Applikationen lassen sich entsprechend dem verfolgten Produktgedanken als die Kernprodukte interpretieren und sind auch weiterhin mit traditionellen Vorgehensweisen zu entwi-

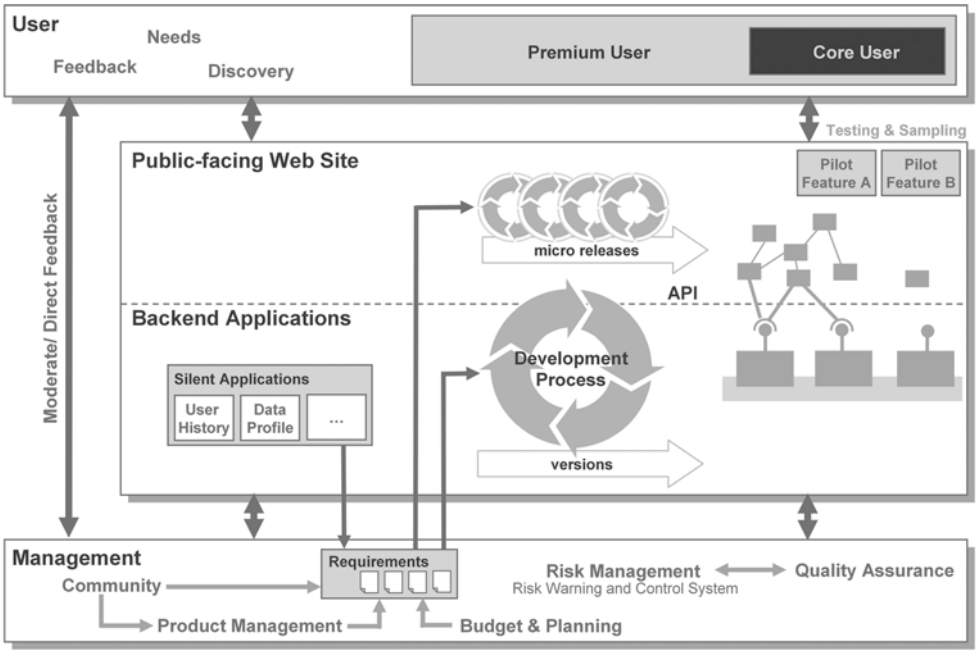

Abb. 3: Web 2.0-Serivceentwicklungsmodell

ckeln. Die Schnittstelle hin zu den eigentlichen Nutzern jedoch ist von häufigen Anpassungen an deren individuelle Bedürfnisse geprägt und erlaubt diesen, selbst als Co-Entwickler aktiv zu werden.

Unter Ausnutzung der umfangreichen Möglichkeiten zum Protokollieren von Web-Anwendungen kann ferner das Nutzerverhalten bezüglich der Services untersucht und mit zuvor durch das Management definierten Zielvorgaben verglichen und ausgewertet werden. Diese Art von privaten leichtgewichtigen Protokollapplikationen lassen sich als »Silent Applications« bezeichnen. Sie geben direkt Rückschlüsse auf einen öffentlichen Service und ermöglichen es, Verbesserungspotenziale für das Produktmanagement aufzuzeigen. Die Ergebnisse nehmen dann wieder über das Management Einfluss auf die Ausgestaltung der Anforderungen. Gleichzeitig bilden »Silent Applications« auch die Informationsbasis des integrierten Risikomanagements, das als Frühwarnsystem eine hohe Qualität der Services garantieren soll.

4 Zusammenfassung und Ausblick

In diesem Artikel wurden wesentliche Veränderungen im Bereich der Softwareentwicklung analysiert, denen Anbieter von Web 2.0-Anwendungen gegenüberstehen. Web 2.0-Plattformen bestehen aus Services, die unter Mitwirkung der Benutzer ständig weiterentwickelt werden. Eine dynamische, sich kontinuierlich weiterentwickelnde Plattform befindet sich gleichsam in einer »ewigen Beta-Version« und stellt neue Herausforderungen an das Management der Entwicklung der zugrundeliegenden Software.

Vor allem die Wahl eines geeigneten Vorgehensmodells, das Design der Softwarearchitektur und der zu verwendende Managementansatz sind von den oben beschriebenen neuen Anforderungen betroffen. Klassische Vorge-

hensmodelle der Softwareentwicklung, die von klar definierten abgegrenzten Phasen ausgehen, sind zu schwerfällig, um die notwendige Dynamik der Entwicklung zu ermöglichen. Als Alternative bieten sich agile Ansätze zur Softwareentwicklung an, die eine inkrementelle Erweiterung vorsehen, auf umfangreiche Dokumentation verzichten, die enge Zusammenarbeit mit dem Auftraggeber berücksichtigen und auf ständige Anpassungen ausgerichtet sind. Zusätzlich zu diesen Eigenschaften der agilen Methoden ist darüber hinaus das Vorgehensmodell um eine dauerhafte operative Phase zu erweitern. Gleichzeitig sollten jedoch die mit der Anwendung von agilen Methoden verbundenen Risiken berücksichtigt werden [Keefer 2003]. Dynamische Web 2.0-Plattformen erfordern zudem flexible komponentenbasierte Architekturen, die eine klare Trennung zwischen fluiden Services und den stabileren Backend-Systemen beinhalten.

Eine sich kontinuierlich weiterentwickelnde Plattform erfordert einen interdisziplinären Managementansatz, der technische Aspekte mit den Bedürfnissen der Nutzer nahtlos verbindet. In Kapitel 3 dieses Artikels wurde ein Produktmanagementansatz vorgestellt, der die oben aufgeführten, neuen Anforderungen berücksichtigt, denen sich die Softwareentwicklung in der Web 2.0-Ära gegenübersieht. Zukünftige Arbeiten werden sich mit der umfassenden Detaillierung dieses Konzepts befassen.

5 Literatur

[Abrahmson et al. 2002] *Abrahmson, P.; Salo, O.; Ronkainen, J.; Warsta, J.:* Agile Software Development Methods – Review and Analysis. VTT Publications 478, 2002.

[Highsmith 2002] *Highsmith, J.:* What is Agile Software Development? In: Cross talk. The Journal of Defense Software Engineering, October 2002.

[Hinchcliffe 2006] *http://blogs.zdnet.com/Hinch cliffe/?p=15FirefoxHTML\Shell\Open\Command;* Zugriff am 12.02.2007.

[Högg et al. 2006] *Högg, R.; Meckel, M.; Stanoevska-Slabeva, K.; Martignoni, K.:* Overview of Business Models for Web 2.0 Communities. In: Proceedings of GeNeMe 2006, Dresden, 2006, S. 23-37.

[Jurison 1999] *Jurison, J.:* Software Project Management: The Manager's View. In: Communications of the Association for Information Systems, Vol. 2, Article 17, September 1999.

[Keefer 2003] *Keefer, G.:* Extreme Programming Considered Harmful for Reliable Software Development 2.0, 2003.

[Lowe & Henderson-Sellers 2001] *Lowe, D.; Henderson-Sellers, B.:* Characteristics of Web Development Processes. In: Proceedings of SSGRR-2001: Infrastructure for E-Business, E-Education, and E-Science, 2001.

[Musser & O'Reilly 2006] *Musser, J.; O'Reilly, T.:* Web 2.0 – Principles and Best Practices. O'Reilly Media, 2006.

[O'Reilly 2005] O'Reilly, T.: What is Web 2.0; *www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html*; Zugriff am 07.02.2006.

[Trittmann et al. 1999] *Trittmann, R.; Hierholzer, A.; Stelzer, D.; Mellis, W.:* Veränderungen der Softwareentwicklung: Ergebnisse einer Fallstudie in der SAP AG im Vergleich zu den Empfehlungen der ISO 9000. Verfügbar online: *www.systementwicklung.uni-koeln.de/forschung/artikel/dokumente/zuerich99.pdf*.

[Zarnekow et al. 2006] *Zarnekow, R.; Brenner, W.; Pilgram, U.:* Integrated Information Management. Springer-Verlag, Berlin, Heidelberg, 2006.

Dipl.-Wirt.-Inf. Volker Hoyer
Dipl.-Ing. Christoph Schroth
Dipl.-Wirt.-Inf. Till Janner
SAP Research CEC
Universität St. Gallen
Institut für Medien und Kommunikationsmanagement
Blumenbergplatz 9
CH-9000 St. Gallen
{volker.hoyer, christoph.schroth, till.janner}@unisg.ch
www.mcm.unisg.ch

Prof. Dr. Katarina Stanoevska-Slabeva
Universität St. Gallen
Institut für Medien und Kommunikationsmanagement
Blumenbergplatz 9
CH-9000 St. Gallen
katarina.stanoevska@unisg.ch
www.mcm.unisg.ch