T. Gross (Hrsg.): Mensch & Computer 2007: Konferenz für interaktive und kooperative Medien. München: Oldenbourg Verlag. 2007, S. 19

Szenariotechniken & Agile Softwareentwicklung

Hartmut Obendorf, Matthias Finck

Zentrum für Architektur und Gestaltung von IT-Systemen, Universität Hamburg

Zusammenfassung

Mit der Erfahrung, dass sich für viele Softwareprojekte erst im Laufe der Entwicklung herausstellt, welche Anforderungen wirklich bestehen, wächst in der Softwaretechnik die Beliebtheit agiler Methoden. Sie versprechen durch den agilen Prozess gegenüber Methoden des Usability-Engineering trotz vergleichbarem Fokus auf Nutzer und Kontext deutlich schneller den Stand ausführbarer Software zu erreichen. Die häufig Feature-getriebene Aushandlung von Anforderungen birgt jedoch das Risiko, den Überblick über den Kontext und damit die einheitliche Vision von Software und ihrer Nutzung zu verlieren. Um dem zu begegnen, schlagen wir die Integration von Szenariotechniken vor. Sie versprechen die Schwächen auszugleichen, die durch die Funktionalitätsfokussierung von XP in der Anforderungsermittlung auftreten können, ohne dabei die Agilität des Prozesses einzuschränken – ein zentrales Problem bisheriger Integrationsansätze.

1 Einleitung

Die Behauptung, dass Usability-Engineering stets von einer möglichst vollständigen Erfassung der Wünsche zukünftiger Nutzer ausginge, und damit gezwungenermaßen von einer gewissen Schwerfälligkeit sei, wäre in dieser Schärfe gewiss ungerecht. Dennoch: in den letzten 10 Jahren hat in der Softwaretechnik unter dem Begriff Agilisierung eine Perspektivenverschiebung stattgefunden, die so noch nicht Eingang in die Softwareergonomie gefunden hat: Inzwischen empfiehlt nicht nur die Standish-Group die Verwendung *agiler Methoden*, auch im neuen V-Modell XT werden explizit agile Vorgehensweisen unterstützt und in der akademischen Diskussion sind agile Methoden fest etabliert (vgl. Boehm 2006).

Durch den starken Fokus auf Nutzer und deren Handeln im Kontext existiert eine enge Beziehung zwischen agilen Methoden und Methoden des Usability-Engineering. Deshalb gibt es inzwischen erste Bestrebungen, Ansätze der jeweils anderen Perspektive zu integrieren (u. a. Kohler et al. 2003; Gundesweiler et al. 2004; Seffah et al. 2005). Dies scheint zunächst einfach: Methoden des Usability-Engineering ermöglichen eine vollständige Modellierung des Systems unter Beteiligung von Nutzern, sodass agile Prozesse direkt von der Erfahrung

in diesem Bereich profitieren können. Da es durch die umfangreiche Analysephase vieler Methoden des Usability Engineering sehr lange dauern kann, bis ein erster Prototyp zum Einsatz kommt, führen die bisherigen Bemühungen der Kombination beider Perspektiven allerdings unmittelbar zu einer „Entagilisierung“ der Methoden – auch wenn teilweise versucht wird, diese Nachteile abzuschwächen (Holtzblatt et al. 2004).

Wir wollen in unserem Beitrag zunächst erläutern, an welcher Stelle die Berücksichtigung von Methoden aus dem Usability-Engineering in agilen Methoden sinnvoll ist, um dann mit Szenariotechniken gezielt eine Methodik herauszugreifen, die die Schwächen agiler Methoden auffängt, ohne deren Agilität zu neutralisieren. Wir stellen ein Prozessmuster vor, das zwei konkrete Ansätze verbindet und berichten über erste Erfahrungen mit dessen Einsatz.

2 Eigenschaften und Defizite agiler Methoden

Agile Entwicklungsmethoden entstanden aus der in der Praxis offenkundig gewordenen Notwendigkeit heraus, mit hoher Flexibilität auf Änderungen in Anforderungskatalogen reagieren zu können, ohne dabei die Qualität der erzeugten Software nachteilig zu beeinflussen. Die Stärke agiler Methoden liegt in ihrer leichteren Planbarkeit, ihr Fokus in der Entwicklung von Software für „organisatorische Systeme“ (Sommerville 2006), also Softwaresysteme, die bedingt durch ihre Einbettung in der realen Welt und die resultierende Wechselwirkung von Entwicklung und Einsatz von einer inkrementellen Entwicklung am stärksten profitieren, da eine vollständige und korrekte Spezifikation der benötigten Funktionalität a priori kaum möglich ist. Bekannte Einschränkungen agiler Methoden sind etwa die mangelnde Skalierbarkeit (Boehm 2006) und die fragliche Anwendbarkeit bestimmter Techniken für Großprojekte (vgl. Stephens & Rosenberg 2003). Zudem wird die vertretene Vorstellung der Einbindung von Endbenutzern als unrealistisch eingeschätzt (vgl. Boehm & Turner 2003, 32) – ein wichtiges Argument für die Kombination mit Methoden des Usability-Engineering.

Ein zentrales Element agiler Prozesse ist ihr *Wesen der Leichtigkeit* (Conboy & Fitzgerald 2004), die Zurückstellung formaler Definitionen der Anforderungen und strikter Anwendungen formalisierter Vorgehensmodelle zugunsten eines stärkeren Fokus auf die direkt am Prozess Beteiligten und die Software selbst. Der Code, und damit die Funktionalität der Software, werden dadurch zum bestimmenden Element für ein Projekt. Diese Konzentration auf benötigte bzw. bereits implementierte Funktionalität wird noch verstärkt von den Aushandlungsprozessen, durch die sich agile Methoden ebenfalls auszeichnen: im Gegensatz zu einer vollständig a priori definierten Liste von Zusicherungen versuchen agile Methoden die Kunden zu einer Priorisierung der Funktionalität zu bewegen und damit Abwägungen im Prozess einvernehmlich mit den Kunden zu treffen. Endbenutzer sind an diesen Aushandlungen jedoch nicht beteiligt – die Kommunikation findet zwischen Budgetverantwortlichen und Entwicklern statt. Die beabsichtigte „Einbeziehung der Benutzer“ ist in den verbreiteten agilen Methoden nicht genau definiert bzw. in unrealistischer Form projektiert (z. B. der onsite customer von XP) – mit Konsequenzen für die Gebrauchsqualität.

Ein Grund für diese Defizite agiler Methoden aus der Perspektive des Usability-Engineering ist in ihrer Stärke begründet, den Fokus auf Einzelheiten zu legen. Das Beispiel *Feature-Driven Development* (FDD) verdeutlicht dies: eine „feature list", die mit hohem Detaillierungsgrad Funktionen des geplanten Systems definiert, wird als zentrales Planungsinstrument sequentiell abgearbeitet (de Luca 2002) und damit versucht, den Verwaltungsaufwand zu reduzieren. In der Folge wird darauf verzichtet, die Anforderungen der Klienten zu hinterfragen bzw. die Kernfunktionalität des Systems herauszuarbeiten. Die iterative Entwicklung, „feature by feature", erschwert so ein einheitliches Design, da an keiner Stelle im Prozess ein vollständiger Entwurf des späteren Systems vorgesehen ist und im Designprozess einzelne Funktionalitäten in das Gesamtbild eingefügt werden müssen – *der Blick aufs Ganze* geht verloren. Dies kann dazu führen, dass Features sich nach ihrer Fertigstellung als unsinnig herausstellen oder zu einer inkonsistenten Bedienung führen. FDD ist dabei kein Einzelfall. Auch andere agile Prozesse vermeiden holistische Entwürfe – es gibt eine Scheu vor „big up-front designs" (Beck 2004), da die Entwicklung von Arbeitsabläufen und organisatorischen Prozessen schwierig, langwierig und nur schwer in ihren Kosten kalkulierbar ist. Stattdessen propagieren agile Methoden die Erstellung eines kleinen Kernsystems und eine kontinuierliche Änderung des Systems in Anpassung an die sich ändernden Anforderungen.

Eine Kopplung des Prozessfortschrittes an einzelne „Features" birgt so die Gefahr, nicht nur den Gesamtentwurf (und damit die Priorisierung einzelner Funktionen) zu erschweren, sondern darüber hinaus die wiederkehrenden *Änderungswünsche* – eine Hauptmotivation für die Entwicklung agiler Methoden – *zum Teil erst zu provozieren*. Im Folgenden beschreiben wir, wie Szenariotechniken mehr Übersicht schaffen und so diese Problematik abmildern können.

3 Szenariotechniken im Usability Engineering

Im Methodenrepertoire des Usability-Engineering (UE) spielen Szenarien eine große Rolle (vgl. Greenbaum & Kyng 1991; Bødker & Christiansen 1997; Carroll 2000). Sie werden als Teil der Dokumentation verstanden, über die Entwickler und Nutzer ein gemeinsames Verständnis des Kontextes und des Softwareeinsatzes entwickeln. Szenarien sind informelle, situative Nutzungsbeschreibungen, was sie gleichermaßen konkret wie flexibel macht. Der Umgang mit ihnen erlaubt eine schnelle Reaktion auf die Unbestimmtheit und Dynamik im Entwicklungsprozess (vgl. Carroll 2000) und weist darin eine große Ähnlichkeit mit agilen Methoden auf. Erkannte Schwächen agiler Softwareentwicklungsmethoden erweisen sich als explizite Stärken von Szenarien: Jarke et al. (1998) fassen ihre Funktion zusammen als Beschreibung der Welt in *einem* Kontext mit einer bestimmten *Absicht* und auf *konkrete* Handlungsabläufe fokussierend. Damit stehen nicht (wie beim FDD) einzelne ‚Features' im Zentrum, sondern *Aktivitäten*, die durch (verschiedene) Funktionen unterstützt werden können.

Szenarien treiben aufgrund ihrer für alle Beteiligten verständlichen Dokumentationsform die Kommunikation zwischen Entwickler und Nutzer voran (Rosson & Carroll 2001). Durch eine mögliche Rückkopplung der Szenarien zwischen Entwickler und Nutzer existiert mit den Szenarien die in agilen Methoden schwach ausgeprägte Möglichkeit, die Anforderungen des Klienten zu hinterfragen bzw. die Kernfunktionalität des Systems herauszufinden. Diese

wird auf natürlich sprachlicher Ebene anhand der zentral beschriebenen Handlungsabläufe bestimmt. Ein weiterer Vorteil gegenüber agilen Methoden ist der ganzheitliche Blick, den Szenarien trotz ihrer Ausschnitthaftigkeit in der Beschreibung behalten. Auch wenn meist auf einen konkreten Handlungsablauf fokussiert wird, so umschließt die Beschreibung stets dessen Einbettung in den größeren Zusammenhang (Jarke et al.1998). Die bei agiler Entwicklung in den Feature-Listen verloren gehenden Abhängigkeiten einzelner Funktionalitäten bleiben in dem Gesamtzusammenhang einer Geschichte erhalten. So können Szenarien sowohl die Forderung agiler Methoden nach dem Beginn mit einem kleinen Kernsystem als auch der Wunsch nach einem ganzheitlichen Blick ohne „big up-front design“ befriedigen.

Szenarien erfüllen im professionellen Einsatz sehr unterschiedliche Funktionen (Rolland 1998), eine sorgfältige Auswahl des passenden Szenariotyps ist also notwendig. Szenarien existieren in verschiedenen Formen und können von Benutzer oder Entwickler geschrieben werden (vgl. u. a. Rosson & Carroll 2001). Sie können z. B. genutzt werden, um als *Modell* den Ist- bzw. Soll-Zustand abzubilden oder als *Medium* zur Exploration (Beyer & Holtzblatt 1997) bzw. der Kooperation zwischen Nutzer und Entwickler dienen (Carroll et al. 1998).

Die genannten Schwächen agiler Methoden legen die Verwendung zweier Szenariovarianten in der Analysephase nahe: *Ist-Szenarien* können Probleme in ihrem Gesamtzusammenhang erfassen und wiedergeben. Es entsteht eine Dokumentation der Anforderungen, die zugleich umfassend und ausreichend unscharf ist, um im Detail flexibel zu bleiben. Zu einem frühen Zeitpunkt wird so ein ganzheitliches Bild der Entwicklung ausgehandelt, das eine reflektierte Aushandlung von Kernanforderungen ermöglicht. *Soll-Szenarien* beinhalten die Beschreibung konkreter Aktivitäten von Nutzern des künftigen Systems, wodurch sich dessen Kernfunktionalität ermitteln und durch weitere Szenarien sukzessive ausweiten lässt. Beide Szenarioarten sollten von Entwicklern geschrieben werden und deren Vorstellungen mit den Nutzern rückkoppeln. Im Gegensatz zum Einsatz vollständiger UE-Prozesse erlauben Szenarien ein schnelles und agiles Vorgehen, begünstigen dabei aber eine holistische Perspektive.

4 Verknüpfungspunkte: Szenariotechniken und XP

Als Diskussionsbasis wurde hier XP gewählt – aufgrund seiner Verbreitung, aber auch wegen der erkennbaren Bestrebungen, Nutzer stärker in den Prozess einzubeziehen; mit der zweiten Auflage von Becks *Extreme-Programming Explained* (2004) verändert sich XP: So wird etwa das kontroverse Konzept des „customer on site“ nicht weiter verfolgt und durch „real user involvement“ ersetzt. Damit wird die Partizipation der Nutzer im Entwicklungsprozess betont, ohne deren Form genau festzulegen. Auch die Rolle des „interaction designer“ wird genannt, ohne diese und ihre Verantwortlichkeiten detailliert zu beschreiben.

Da agile Methoden auf die Optimierung des Entwicklungsprozesses abzielen, ohne dabei ein bestimmtes Gestaltungsmedium vorzugeben, ist dessen Wahl entscheidend für die Art der Gestaltung des entworfenen Produktes. Die Wahl von Szenarien verspräche auf eine Kernfunktionalität zu fokussieren und zugleich den Gesamtentwurf im Auge zu behalten. XP ist ein Prozess, der ausdrücklich formalen Repräsentationen wenig Wert beimisst, und stattdes-

sen auf die inkrementelle Ausgestaltung angeforderter Funktionalität in Interaktion mit dem Kunden setzt. Es sind also Techniken vonnöten, die auch als leichtgewichtig gelten können, gleichzeitig aber unnötige Iterationsschritte vermeiden helfen. XP beinhaltet mit seiner *story*-Technik bereits eine Repräsentation, die Szenarien ähnelt, dabei aber weniger Kontextinformation beinhaltet (vgl. Cohn 2004). Für eine Integration kommen daher nur Szenariotechniken in Frage, die weniger Wert auf Kompatibilität mit Entwurfstechniken wie UML legen (wie etwa Ambler & Jeffries 2006), dafür aber einfach von Entwicklern wie Nutzern verstanden werden können. Die Granularität der gewählten Szenarien sollte einerseits sehr grob sein, um eine ganzheitliche Beschreibung der Systemnutzung zu erlauben; andererseits sollte die Möglichkeit bestehen, ausgewählte Funktionalität sehr detailliert zu beschreiben.

Rosson und Carroll's *Scenario-Based Design* (SBD 2001) stellt ein Vorgehensmodell dar, das in allen Phasen des Entwicklungsprozesses auf die Erstellung von Szenarien setzt, indem zunächst Problemszenarien den Ist-Zustand beschreiben, eine Vision und Aktivitätsszenarien den groben Systementwurf festhalten, und dieser dann inkrementell bis auf Entwurfsebene immer wieder in Szenarioform verfeinert wird. Für die Integration in den XP-Prozess bieten sich drei Szenarioformen an: *(1) Problemszenarien* mit Fokus auf die Anwendungsdomäne, *(2) Vision + Aktivitätsszenarien*, die grundsätzliche Entwurfsgedanken festhalten, und *(3) Storyboards bzw. Papierprototypen*, die XP-Stories kontextualisieren und konkretisieren.

4.1 Die Schnittstellen von XP- und Szenariotechniken

Die Einbeziehung der Kontextanalyse in Form von Problemszenarien erfordert ein methodisches Vorgehen bei der Erforschung des Aufgabenkontexts. Dafür bietet sich als Anhaltspunkt das *Contextual Inquiry* (vgl. Beyer et al. 1997) an: Entwickler besuchen in Paaren oder zu Dritt Endbenutzer an Ihrem Arbeitsplatz, wo sie sich von diesen typische und wichtige Arbeitsabläufe erklären lassen; dabei ist es wichtig, dass ein Besucher die Rolle eines „Lehrlings" einnimmt, während die Begleiter im Hintergrund eine möglichst konzise Kontextdokumentation in Notizen, Skizzen und Fotos anfertigen. Durch diese Meister-Lehrling-Beziehung zwischen Benutzer und Entwickler steht die Expertise des Endanwenders im Mittelpunkt der Erkundung, die Entwickler werden aus der Rolle der wissenden Analytiker herausgedrängt und die Arbeitsabläufe werden „sichtbar gemacht" (Suchman 1995).

Im Folgenden werden gesammelte Eindrücke konsolidiert und ein *Problemszenario* (vgl. Rosson & Carroll 2001) erstellt, das nur jene Arbeitsabläufe detaillierter beschreibt, die von den Entwicklern als zentral identifiziert wurden. Die gezielte und unmittelbare Rückkopplung ermöglicht, eine falsche Fokussierung der Aufgabenanalyse frühzeitig zu erkennen und diese iterativ zu verbessern. Nachdem auf diese Weise die Kernaufgaben der zu entwickelnden Software identifiziert worden sind, versuchen die Entwickler, ggf. in Kooperation mit einzelnen Benutzern, eine Lösung zu finden, die gleichzeitig das identifizierte Problem möglichst optimal löst, sich andererseits aber auch problemlos in dem im Problemszenario beschriebenen Arbeitskontext einbetten lässt, also z. B. mit anderen benutzten Werkzeugen harmoniert bzw. ähnliche Konzepte nutzt. Diese Lösungsidee wird in einem kurzen Leitsatz, der *Vision*, festgehalten, die den Charakter der Herangehensweise beschreibt (z. B. mit Hilfe einer Metapher). Konkretisiert wird der Lösungsansatz durch eine Beschreibung, wie die im Problemszenario beschriebenen bisherigen Arbeitsabläufe mit Hilfe der zu entwickelnden

Lösung bearbeitet werden können, dem *Aktivitätsszenario* (entsprechend dem Soll-Szenario). Dies geschieht ausschnittweise anhand einzelner konkreter Aktivitäten, entwirft aber wieder ein Gesamtbild der nun veränderten Nutzung im Arbeitskontext.

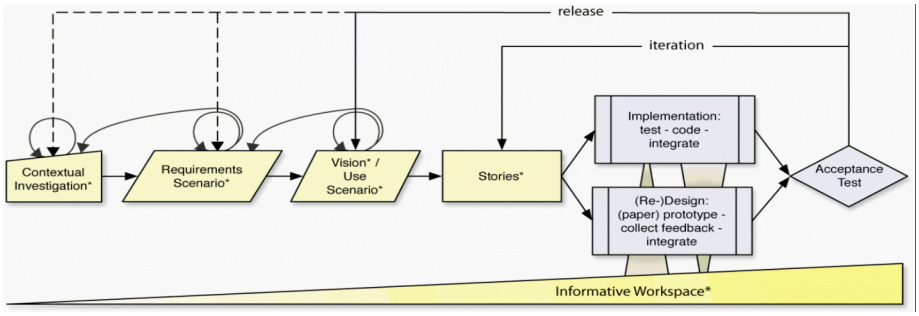

Abbildung 1: Schematische Darstellung der Schnittstellen von XP- und Szenariotechniken.

XP verlässt sich darauf, dass der Kunde die Spezifikation für die Funktionalität der nächsten Iterationsschritte liefert. Dabei spielen *stories*, textuelle Kurzbeschreibungen, eine zentrale Rolle in der Kommunikation zwischen Kunde und Entwickler. Um die Aushandlung auf die Gestaltung der Arbeitsabläufe und das geplante Verhalten des Systems auszuweiten, müssen diese Stories erweitert werden: Durch Kontextualisierung wird die Aushandlung von Funktionalität in Bezug zur Gesamtheit der Lösungsidee gesetzt, und durch Konkretisierung – die z. B. mit Papierprototypen (vgl. Snyder 2003) eine schnellere Rückkopplung ermöglicht – werden für wichtige Designentscheidungen frühzeitig möglich, die Softwarearchitektur kann gleichberechtigt auf technischen wie fachlichen Anforderungen entwickelt werden. Storyboards (vgl. z. B. Beyer et al. 1997) können dabei als Elemente einer Story einbezogen werden: Illustrationen vom Einsatzkontext einer Funktionalität, und Entwürfe der Interaktion mit dieser bilden die Grundlage für eine Diskussion mit den Nutzern und gleichzeitig eine Dokumentation, auf der die Entwicklung aufsetzen und inkrementell wichtige Detailfragen klären kann, die dabei mit der gemeinsam gefundenen Lösungsidee vereinbar bleiben.

Der modifizierte XP-Prozess ist weiterhin als in hohem Maße *iterativ* zu verstehen. Die vorgeschaltete Anforderungsermittlung ist bewusst *unvollständig*, in späteren Phasen werden deshalb weitere Erkundungsgänge nötig, und weitere Designentscheidungen müssen getroffen werden. Da die Notwendigkeit, eine agile Lösung zu finden, die kontinuierliche Nachführung von Szenarien als Entwurfsdokumentation, oder auch die Erstellung eines vollständigen Archivs der Kontexterkundungen nicht zulässt, muss auch der „informative workspace“ von XP stark erweitert werden: während er weiterhin die Funktion hat, den Projektfortschritt zu visualisieren und organisieren, übernimmt ein Teil des *workspace* die Aufgabe, das Verständnis des Teams von der Problemdomäne und den anvisierten Lösungen zu unterstützen. Ähnlich dem „affinity diagram“ aus dem *Contextual Design* (vgl. Beyer et al. 1997; Holtzblatt et al. 2004) werden dabei wichtige Aspekte der Arbeitsgestaltung so gruppiert, dass der Bezug von Problemsituationen und Lösungsansätzen räumlich offenbar wird; alte Lösungsvorschläge bleiben so lange erhalten, bis sie obsolet oder überarbeitet sind.

5 Erfahrungen mit dem integrierten Vorgehen

Um die Auswirkungen den entwickelten Prozessmusters besser einschätzen zu können, wurde es im Rahmen eines Projektseminars am Department Informatik der Universität Hamburg im Sommersemester 2005 angewendet. Die Veranstaltung wurde mit 18 Studierenden und 3 Lehrenden der Softwaretechnik und Softwareergonomie durchgeführt, so dass praktische Erfahrung in Anwendung und Vermittlung von XP (vgl. Becker-Pechau et al. 2003) und mit Szenariotechniken (z.B. SBD) bestand. Ziel war es, Probleme der Kombination der verschiedenen Methoden herauszuarbeiten; die regelmäßige gemeinsame Reflektion der eingesetzten Methoden dient hier zusammen mit einer formativen Umfrage als Grundlage der Diskussion.

Als Anwendung wurde ein verteiltes Kalendersystems zur Unterstützung des Universitätspersonals gewählt – bevor die Entwicklung beginnen konnte, musste das Team also verstehen, welche Aufgaben ein Kalender in der täglichen Arbeit von Verwaltungspersonal und wissenschaftlichen Mitarbeitern erfüllen muss. Mit dieser umfangreichen und realistischen Problemstellung gelang es, eine Reihe praxisnaher Probleme zu erzeugen – wie Situationen erheblichen zeitlichen Drucks, Konfrontation mit realen Nutzern und ihren Anforderungen und Kapazitätsengpässen auf Seiten der Nutzer wie Entwickler. Nach zwei Entwicklungszyklen entstand dennoch ein lauffähiger Prototyp, der die Kernfunktionalität des verteilten Kalendersystems vollständig enthielt. Auf einen Vergleich mit „XP pur“ verzichteten wir, da unserer Erfahrung nach in realen Projekten XP abhängig von Team und Aufgabenstellung unterschiedlich implementiert und allein dadurch schon eine starke Variation einführt wird.

Viele Studenten berichteten, dass das Schreiben der Problemszenarien eine der größten Herausforderungen für sie war. Die Probleme bei der Erstellung lagen vor allem in den unterschiedlichen Verwendungszwecken von Szenarien, je nach dem, ob sie aus softwaretechnischer oder software-ergonomischer Perspektive erstellt wurden. Die Studenten, die über weit mehr Wissen und Praxiserfahrung im Bereich XP verfügten, konzentrierten sich in einer ersten Fassung der Szenarien stark auf die Schilderung der Verwendung konkreter Features und der daraus resultierenden Reaktion des Systems – eine sehr systemtechnische Sicht (vgl. Hitz et al. 2005), die es zunächst erschwerte, die Problematik eines verteilten Kalenders und damit eine ganzheitliche Systemvision zu erarbeiten. Nach einer Reflexion über die Art der erstellten Szenarien und einer Überarbeitungsphase konnten die Szenarien dann – wie geplant – genutzt werden, um die zentralen Handlungen mit existierender Software bzw. Papierkalendern zu identifizieren. Im Nachhinein beurteilten gerade die Studenten mit dem größeren Erfahrungshintergrund den Einsatz der stärker nutzungs- als systemzentrierten Szenarien als besonders hilfreich im Prozess. Aus unserer Sicht liegt ein Grund darin, dass die Szenarien eine subtile Leitungsfunktion übernommen haben; obwohl das Gesamtteam sich in der Folge in mehrere Untergruppen aufteilte, die z. T. völlig schnittstellenferne Programmierarbeiten ausführten, war allen Teilnehmern ein ungewöhnlich starkes Bewusstsein für das gemeinsame, an Nutzeraktivitäten ausgerichtete Ziel anzumerken.

Dies wird von den Ergebnissen des Gestaltungsprozesses insofern gestützt, als das entwickelte Kalendersystem wirklich auf einen Anwendungsfall fokussiert, ohne dabei an allgemeiner Einsatzfähigkeit zu verlieren: auch wenn Kalender mehrerer Nutzer dargestellt wer-

den, bleibt die Übersicht erhalten (Abb. 2) – die zusätzlichen Kalender werden, ähnlich wie bei geografischen Informationssystemen, wie Folien über die aktuelle Sicht gelegt. Dieses Design wurde durch eine Konzentration auf die in den Szenarien beschriebene Kernaktivität gefunden: der Suche nach freien Zeiteinheiten, um Gruppentermine vereinbaren zu können.

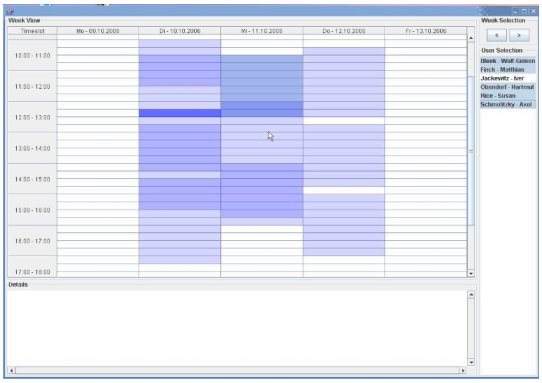

Abbildung 2: Der entwickelte Prototyp mit Folienlayout zur Terminfindung: eine einfache, aber effektive Lösung.

Erste Tests mit Nutzern ermittelten eine sehr positive Aufnahme des Prototyps – auch im Vergleich zu existierenden Lösungen. Besonders hervorzuheben ist, dass das initiale, deutlich komplexere Design zugunsten einer schlichteren Variante verworfen wurde – nicht durch die Veranstalter gelenkt, sondern ausschließlich durch den Rückbezug auf die in den Szenarien geschilderten Kernaktivitäten bedingt. Die Studenten entschieden, dass es, wenn die wichtigste Aktivität das Finden eines freien Termins ist, nicht darauf ankommt, erkennen zu können, wer und aus welchen Gründen keine Zeit hat; damit konnte auf die Lesbarkeit der übereinander liegenden Termine verzichtet werden.

6 Diskussion

Bisher existiert keine zufrieden stellende Lösung, die die technischen Vorteile agiler Entwicklungsmethoden mit dem Fokus auf Gebrauchsqualität, der vom Usability Engineering eingebracht wird, vereinen kann (Birk et al. 2003) – bisherige Gehversuche handelten sich mit einer extensiven Phase der Anforderungsermittlung den Verlust von Agilität und iterativem Vorgehen ein (vgl. z. B. Gundelsweiler et al. 2004), oder beschränkten sich auf minimale Veränderungen, etwa durch Discount Usability Engineering (z. B. Kane 2003).

Der in diesem Beitrag vorgeschlagene Weg führt über die Einbeziehung verschiedener Szenariotechniken in den agilen Entwicklungsprozess zu einer Prozessgestaltung, die sich als gleichermaßen agil wie partizipativ erweist, indem sowohl die allgemeine Gestaltungsidee als auch konkrete Lösungsdetails immer wieder direkt mit den Benutzern rückgekoppelt werden können – schneller und umfassender, als dies mit den Mitteln von XP möglich ist.

Szenarien spielen dabei eine Schlüsselrolle in der Fokussierung der Problemstellung und der Entwicklung einer Lösungsidee; in unserem Beispielprozess wurde die Problematik, mit mehreren, sehr beschäftigten Personen einen Termin zu vereinbaren, als Kernfunktionalität definiert. Durch die Szenarien wurde es möglich, eine Lösung zu finden, die unvollständig und zum Teil nur grob skizziert war, aber die Gesamtheit der Problemstellung mit einbezog und einfach ausgebaut werden konnte. Diese Eigenschaft von Szenarien, eine abstrakte Gesamtvision zu dokumentieren, die dann auf spezielle Teillösungen sehr konkret übertragen werden kann, begründet eine wesentliche Stärke des vorgestellten Vorgehens.

Szenarien unterstützen so die iterative Entwicklung, die ein wesentliches Merkmal agiler Methoden darstellt, und verknüpfen die Arbeit an Detailfragen mit einer Gesamtsicht auf das zu entwickelnde System. Die Anwendung von Szenariotechniken verspricht also, Schwächen auszugleichen, die durch den auf Funktionalität fokussierten Ansatz von XP in der Anforderungsermittlung auftreten können – ohne dabei größere Einschränkungen der Agilität des Entwicklungsprozesses zu bewirken; durch Verwendung von Szenarien als Prototyping-Methode lässt sich diese in bestimmten Bereichen sogar noch erhöhen.

Der Einsatz von Szenarien in realen agilen Entwicklungsprojekten wird zurzeit vorbereitet bzw. erprobt. Die Form des Einsatzes wird zwar – wie stets auch die konkrete Durchführung agiler Techniken – von Projekt zu Projekt unterschiedlich ausfallen, wir versprechen uns aber dennoch Erkenntnisse über eine Verallgemeinerbarkeit unser bisherigen Erfahrungen.

Literatur

Ambler, S. W.; Jeffries, R. (2002). Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. Wiley.

Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A. et al. (2001). Agile Manifesto. Verfügbar unter http://www.agilemanifesto.org (letzter Zugriff 20.2.2007).

Beck, K. (2004). Extreme Programming Explained: Embrace Change, 2. Auflage, Addison-Wesley.

Becker-Pechau, P.; Breitling, H.; Lippert, M.; Schmolitzky, A. (2003). Teaching Team Work: An Extreme Week for First-Year Programmers. Extreme Programming and Agile Processes in Software Engineering, 4th International Conference, XP 2003, S. 386-393.

Beyer, H. & Holtzblatt, K. (1997). Contextual Design : A Customer-Centered Approach to Systems Designs. Morgan Kaufmann.

Birk, A.; Kohler, K.; Leidermann, F. (2003). Der Weg zu einer stärkeren Verzahnung von Usability Engineering und Software Engineering. UPA-Workshop Mensch & Computer 2003.

Boehm, B. (2006). A view of 20th and 21st century software engineering. Shanghai, S. 12–29.

Boehm, B.; Turner, R. (2003). Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley Professional.

Bødker, S.; Christiansen, E. (1997). Scenarios as springboards in the design of CSCW. In Social science research, technical systems and cooperative. Mahwah, NJ: Erlbaum. S. 217-234.

Carroll, J. M.; Chin, G.; Rosson, M. B.; Neale, D. C. (2000). The development of cooperation: five years of participatory design in the virtual school. DIS '00: Proceedings of the conference on Designing interactive systems, New York, NY, USA, S. 239-251.

Carroll, J. M.; Rosson, M. B.; Jr, George Chin; Koenemann, J. (1998). Requirements Development in Scenario-Based Design. IEEE Trans. Softw. Eng, 24(12), S. 1156-1170.

Cohn, M. (2004). User Stories Applied: For Agile Software Development. Addison Wesley.

Conboy, K.; Fitzgerald, B. (2004). Toward a conceptual framework of agile methods: a study of agility in different disciplines. Newport Beach, CA, USA, S. 37–44.

de Luca, J. (2002). Latest FDD processes. Verfügbar unter http://www.nebulon.com/articles/fdd/download/fddprocessesA4.pdf (letzter Zugriff 20.2.2007).

Greenbaum, J., Kyng, M. (1991). Design at Work: Cooperative Design of Computer Systems. Lawrence Erlbaum Associates Inc.

Gundelsweiler, F.; Memmel, T.; Reiterer, H. (2004). Agile Usability Engineering. Mensch & Computer 2004: Allgegenwärtige Interaktion, München, S. 33-42.

Hitz, M.; Kappel, G.; Kapsammer, E.; Retschitzegger, W. (2005). UML@Work. Objekt-orientierte Modellierung mit UML2. Heidelberg: d-punkt Verlag.

Holtzblatt, K.; Wendell, J. B.; Wood, S. (2004). Rapid Contextual Design: A How-to Guide to Key Techniques for User-Centered Design. Morgan Kaufmann.

Jarke, M.; Bui, X. T.; Carroll, J. M. (1998). Scenario Management: An Interdisciplinary Approach. Requirements Engineering Journal, 3(3-4).

Kane, D. (2003). Finding a Place for Discount Usability Engineering in Agile Development: Throwing Down the Gauntlet. ADC '03: Proceedings of the Conference on Agile Development, S. 40ff.

Kohler, K.; Leidermann, F.; Birk, A. (2003). Der Weg zu einer stärkeren Verzahnung von Usability Engineering und Software Engineering. In Usability Professionals 2003. Stuttgart, S.21-25.

Rolland, C.; Ben Achour, C.; Cauvet, C. et al. (1998). A Proposal for a Scenario Classification Framework. Requirements Engineering Journal, 3(1), S. 23-47.

Rosson, M. B.; Carroll, J. M. (2001). Usability Engineering Scenario-based Development of Human Computer Interaction. Morgan Kaufmann.

Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. Monterey, California, United States, S. 328–338.

Seffah, A., Gulliksen, J., Desmarais, M. C. (2005). Human-Centered Software Engineering – Integrating Usabilty in the Software Development Lifecycle, Springer.

Snyder, C. (2003). Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces. Morgan Kaufmann.

Sommerville, I. (2006). Software Engineering (8th Edition). Addison Wesley.

Stephens, M.; Rosenberg, D. (2003). Extreme Programming Refactored: The Case Against XP. Apress.

Suchman, L. (1995). Making work visible. Commun. ACM, 38(9), S. 56ff.