Erschienen in: Mensch und Computer 2004 : Allgegenwärtige Interaktion / Keil-Slawik, Reinhard et al. (Hrsg.). - München : Oldenbourg, 2004. - S. 33-42. - ISBN 3-486-57624-0
http://dx.doi.org/10.1524/9783486598773.33

R. Keil-Slawik, H. Selke, G. Szwillus (Hrsg.): Mensch & Computer 2004: Allgegenwärtige Interaktion. München: Oldenbourg Verlag. 2004, S. 33–42

Agile Usability Engineering

Fredrik Gundelsweiler, Thomas Memmel, Harald Reiterer

Universität Konstanz, Informatik & Informationswissenschaft,
Arbeitsgruppe Mensch-Computer Interaktion

Zusammenfassung

Der Beitrag untersucht die Fragestellung, ob die Prinzipien, Methoden und Techniken agiler Softwareentwicklungsprozesse mit denen des Usability Engineering vereinbar sind. Um diese Frage zu beantworten, wurden vor allem die Praktiken von Extreme Programming und Agile Modeling näher untersucht und dann vom Standpunkt der Praktiken des Usability Engineering aus bewertet. Die Autoren kommen zu dem Schluss, dass es neben grundlegenden Zielkonflikten auch eine Reihe von interessanten Anknüpfungspunkten gibt, und stellen als eine mögliche Synthese aus beiden Welten den „Agile Usage-Centered Software Lifecycle“ (AUCSL) vor. Damit soll eine Diskussion in der Fachwelt angeregt werden, verbunden mit der Erwartung einer praktischen Erprobung.

1 Einleitung

Die in diesem Beitrag untersuchte Fragestellung lautet: „Stehen die Prinzipien, Methoden und Vorgehensweisen eines agilen Softwareentwicklungsprozesses im Widerspruch zu den gängigen Prinzipien, Methoden und Vorgehensweisen des Usability Engineering (UE)?“ Die Fragestellung entstand im Rahmen einer Kooperation der Arbeitsgruppe Mensch-Computer Interaktion der Universität Konstanz mit der Forschungsabteilung der Firma DaimlerChrysler AG in Ulm. Da agile Methoden der Softwareentwicklung (SE) in den letzten Jahren – zumindest wenn man der einschlägigen Literatur folgt (Cockburn 2001; Ag-ileAlliance 2004) – immer mehr an Beachtung gefunden haben, stellt sich die Frage, ob die damit verbundenen Vorgehensweisen mit denen des UE vereinbar sind. Von den vielen agilen Methoden wurde der Fokus auf Extreme Programming (XP) gesetzt. Zur Beantwortung unserer Fragestellung wurde zuerst eine umfassende Literaturstudie zum Thema XP und in weiterer Folge zu Agile Modeling (AM) durchgeführt. Die Ergebnisse dieser Recherchen haben wir dann im Lichte unserer Kenntnisse im Bereich UE bewertet (kurze Auswahl besonders einschlägiger Quellen: Beyer & Holzblatt 1998; Constantine & Lockwood 1999; Mayhew 1999; Rosson & Carroll 2002; Preece et al. 2002) und folgende Hypothese formuliert: „Der Software Lifecycle von XP, erweitert um AM Praktiken und ergänzt um agile UE Methoden, ermöglicht eine agile Entwicklung gebrauchstauglicher Software.“

Im folgenden Abschnitt wird kurz erläutert, auf welchen Einschätzungen die vorgestellte Hypothese basiert, indem wir Praktiken des XP und AM denen des UE gegenüberstellen. Im anschließenden Abschnitt „Der Agile Usage-Centered Software Lifecycle“ wird ein von uns konzeptionell entwickelter Software Lifecycle vorgestellt, der den Ansprüchen von AM, XP und agilem UE Rechnung trägt.

2 Gegenüberstellung von Prinzipien und Praktiken des XP, Agile Modeling & Usability Engineering

XP (Beck 1999) stellt eine besonders radikale Form der agilen Softwareentwicklung dar. Dennoch stellen wir bereits bei der Betrachtung von XP Gemeinsamkeiten mit Praktiken des UE fest, welche in Tabelle 1 auszugsweise wiedergegeben werden.

XP Praktiken	**Usability Engineering Praktiken**
Iteration, Small Increments, Adaptivity	Prototyping
Story Cards, Task Cards, User Stories	User Profile, Task Model
On-Site Costumer	User-Centered Design, User Participation
Testing, Test Story	Evaluation, Usability Inspections
Metaphor	Conceptual Model, Mental Model, UI Metaphor

Tabelle 1: Ausgewählte XP- und UE-Praktiken

Scott Ambler kommt hinsichtlich XP zu folgender kritischen Einschätzung: *„… although XP clearly includes modeling as part of its process, it is not as explicit about how to do so as many developers would prefer …“* (Ambler 2002, S.177). Wir erweitern deshalb unsere Betrachtungsweise um Praktiken, Methoden und Techniken des AM (vor allem basierend auf Arbeiten von Ambler 2002). Auch beim Vergleich von AM- und UE-Praktiken stellen wir eine Reihe von Gemeinsamkeiten fest. Sie sind in Tabelle 2 wiedergegeben. Dabei ist hervorzuheben, dass insbesondere (Ambler 2002) eine Reihe von agilen UE-Methoden – vor allem basierend auf Arbeiten von (Constantine & Lockwood 1999) – beinhaltet.

Agile Modeling Praktiken	**Usability Engineering Praktiken**
Prove It With Code	Prototyping
Create Several Models in Parallel	Concurrent Modeling (User Role-, Task-, Content-Model)
Active Stakeholder Participation	Usage-Centered Design, User Participation
Consider Testability	Evaluation, Usability Inspections
Display Models Publicly	Design Room

Tabelle 2: Ausgewählte AM- und UE-Praktiken

Sicherlich existieren neben den festgestellten Übereinstimmungen auch grundlegende Konflikte zwischen den Prinzipien und Praktiken. Beispielsweise fordern agile Vorgehensweisen frühe und schnelle Ergebnisse. Deshalb soll so schnell wie möglich mit der Entwicklung der Software begonnen werden (Code Matters!) und der Dokumentations- und Spezifikationsprozess soll möglichst knapp gehalten werden. Dazu wird der Entwicklungsprozess in kleine, überschaubare Problemstellungen zerlegt (Small Releases) und inkrementell ausgeführt. Während der Projektlaufzeit muss es möglich sein, Anpassungen in verschiedenem Umfang vorzunehmen (Refactoring).

Im Gegensatz zu dieser Philosophie wird im UE zu Beginn relativ viel Zeit für die Anforderungsermittlung und die Spezifikation der User Interface Architektur genutzt. In die eigentliche Entwicklung soll erst eingestiegen werden, wenn genügend Wissen bezüglich des Nutzungskontextes vorhanden ist und eine vollständige Architektur des User Interface (UI) (z.B. grundlegendes Navigationskonzept und Interaktionstechniken, grundlegende Organisation der Informationspräsentation) entwickelt worden ist. Dieser klare Zielkonflikt kann mit folgendem Zitat von Constantine (2002, S.5) gut belegt werden:

„The most critical shortcoming of nearly all techniques that are based on iterative expansion and refinement in small increments is the absence of any comprehensive overview of the entire architecture. For internal elements of the software, this shortcoming is not fatal, because the architecture can be defined and restructured at a later time. Refactoring can, in many cases, make up for the absence of a complete design in advance. UI are a different story.“

Begründet wird dies damit, dass ein permanentes Redesign des UI zu massiven Benutzungsproblemen führen kann. So würde eine grundsätzliche Veränderung des Navigationskonzeptes, der Organisation der Informationspräsentation oder der gewählten Interaktionstechniken von Small Release zu Small Release dazu führen, dass grundlegende Dialogprinzipien des UE wie Erwartungskonformität, Selbstbeschreibungsfähigkeit oder Erlernbarkeit verletzt würden. Die agilen Softwareentwicklungsprozesse vernachlässigen aus Sicht des UE eine Reihe wichtiger Vorgehensweisen zur Entwicklung gebrauchstauglicher Software. Dennoch sind wir der Ansicht, dass die Gegenüberstellung der Prinzipien es zulässt, eine Reihe von interessanten Anknüpfungspunkten zu identifizieren, um eine Annäherung zu wagen (z.B. Benutzerorientierung über On-Site-Customer, Aufgabenorientierung über Story und Task Cards, frühes Prototyping dank Small Increments, Evaluationsorientierung dank starker Rolle des Testens, Metaphern als Anknüpfungspunkt für das Conceptual Model). Wir sehen in der agilen SE eine große Chance, da sich wesentliche UE-Prinzipen im Bereich des Software Engineering wieder finden. Damit könnte der vielfach in der Literatur (Dzida & Freitag 1998) und Praxis beklagte fehlende Brückenschlag zwischen beiden Disziplinen gelingen!

Folglich kommen wir zu dem Ergebnis, dass es möglich ist, um AM-Praktiken angereichertes XP und UE in einem gemeinsamen Lifecycle zu vereinen, in dem sich die Stärken der Ansätze verbinden und Nachteile herkömmlicher XP-Vorgehensweisen vermeiden lassen. Der von uns im nächsten Abschnitt vorgestellte „Agile Usage-Centered Software Lifecycle“ erweitert somit die Vorgehensweisen und Praktiken von XP bzw. AM und ermöglicht unter Verwendung der von uns identifizierten agilen UE-Techniken eine agile Entwicklung gebrauchstauglicher Software.

3 Der Agile Usage-Centered Software Lifecycle

Bei agilen Methoden und insbesondere bei XP wird hervorgehoben, dass alle Mitglieder des Projektteams vergleichbare Qualifikationen haben sollten. In einem rein technisch orientierten Projektteam ist es sicherlich wünschenswert, dass die Programmierer auf gleichem Qualifikationsniveau sind. In unserem Softwareprozess gehen wir aber von einem heterogenen Entwicklerteam aus, da unserer Meinung nach auf den Gebieten Grafikdesign und UE Qualifikationen notwendig sind, die auch von einem „agilen" Programmierer nicht zu erwarten sind. Unser Team sollte somit folgende Rollen kennen: Programmierer, Usability-Experte, Designer und Stakeholder. Letztere sollten neben den Auftraggebern, dem Management oder anderen Interessenvertretern vor allem die tatsächlichen Benutzer sein. Des Weiteren sollte ein so genannter „Tracker" die Einhaltung der Usability-Ziele überprüfen. Grundsätzlich ist es auch bei der von uns vorgeschlagenen Teamzusammensetzung nicht ausgeschlossen, dass eine Person mehrere Rollen übernehmen kann (siehe 3.3).

Abbildung 1 zeigt die einzelnen Phasen des AUCSL, der an den XP-Prozess (Beck 1999; EC Wise 2003) angelehnt ist, sich aber in wesentlichen Punkten von diesem unterscheidet.

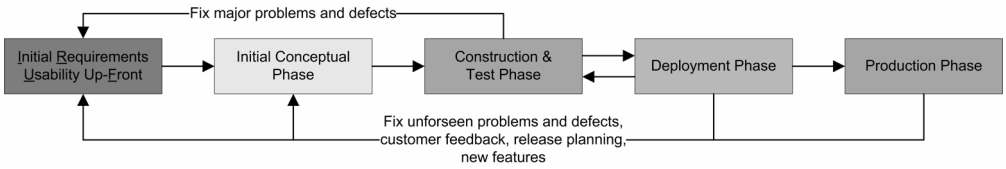

Abbildung 1: Phasen des „Agile Usage-Centered Software Lifecycle" (AUCSL)

3.1 Initial Requirements Usability Up-Front

Unserer Meinung nach reicht eine Betrachtung der Architektur, wie sie in agilen Prozessen vorgesehen ist, nicht aus. Um Software mit einem gebrauchstauglichen UI zu erstellen, ist eine Integration von UE erforderlich. Weder AM noch XP sieht Analysen vor, die für grundlegende UE-Aktivitäten notwendig sind. Grundsätzlich liefern Stakeholder (bei XP der On-Site-Customer) die Anforderungen in Form von User Stories. Problematisch ist, dass die Kunden nicht zwingend die tatsächlichen Benutzer der Software sind. So ergeben sich einige offene Fragen bei der Erstellung von User Stories durch Dritte: Wie würden die tatsächlichen Benutzer handeln und damit die Story schreiben? Woher haben die Befragten das Verständnis für die zu entwickelnde Anwendung? Ist das erwünschte Programmverhalten über alle Anforderungen konsistent? Passt die Story in den Work Flow des Benutzers? Diese Fragen können aus agiler Sicht nur unzureichend beantwortet werden. Die eigentlichen Benutzer bleiben bei der Anforderungsermittlung außen vor. Für ein gutes UI-Design sind Benutzer und ihre Rollen aber unbedingt erforderlich. Methodisch können diese sehr gut mit Role Models und Role Maps (Constantine & Lockwood 1999) erhoben werden. Essentiell sind

auch die Aufgaben der Benutzer, die durch Szenarien, Essential Use Cases und Use Case Maps (Constantine & Lockwood 1999) beschrieben werden können. Der wesentliche Vorteil der Role Models/Role Maps und Essential Uses Cases/Use Case Maps aus agiler Modellierungssicht besteht darin, dass sie gut zur grundsätzlichen Philosophie passen (z.B. Nutzung von Indexkarten, Fokussierung auf das Essentielle).

Wir schlagen mit unserer Initial Requirements Usability Up Front (IRUUF) eine agile, benutzerorientierte Anforderungsermittlung vor.

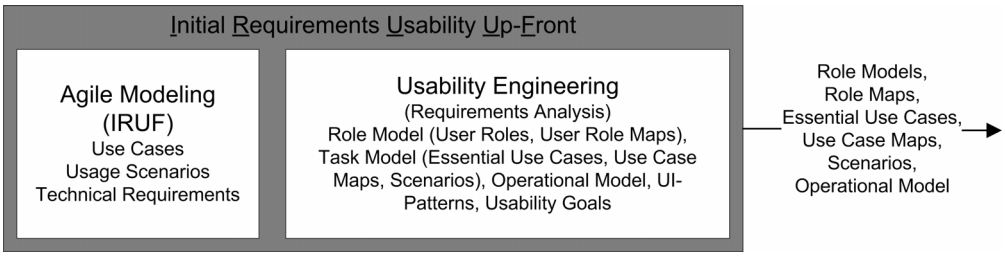

Abbildung 2: Phase „IRUUF" (Initial Requirements Usability Up-Front) des AUCSL

Abbildung 2 zeigt auf der linken Seite die Methoden und Techniken des AM, die um die Techniken des UEs (auf der rechten Seite) ergänzt und teilweise auch ersetzt werden können.

Zur Modellierung der Benutzerrollen mittels agiler Methoden wie Role Models und Role Maps werden zunächst alle denkbaren Rollen der geplanten Software durch Focus Groups mit allen Stakeholdern ermittelt. Dokumentiert werden sollen die für die Anwendung wichtigen Charakteristika der Rollen sowie erforderliche Interaktionsschemata. Anschließend werden die Benutzerrollen mit Prioritäten versehen (Focal User Roles) und nach Relevanz für den Produkterfolg sortiert. Außer den Benutzerrollen werden auch die Szenarien agil erstellt. Im Gegensatz zu den in AM und XP verwendeten Use Cases benutzen wir hier agilisierte Usability-Techniken in Form von Essential Use Cases, Task Maps und Szenarien. Alle Essential Use Cases/Task Cases bekommen einen mit der Aufgabe assoziierten Namen. Wie schon die Benutzerrollen werden auch die Task Cases sortiert, wobei wir uns an der aus XP (Beck 1999) bekannten Einteilung orientieren (Required - do first, Desired - do if time, Deffered - do next time). Ergänzt wird diese Einteilung um essentielle Task Cases, die besonders komplex oder neuartig sind. Diese werden zu Szenarien ausformuliert, um das Verständnis im Team besser zu kommunizieren. Sind die Aufgaben alle erfasst, ist es möglich Cluster zu bilden, die innerhalb eines Anwendungsbereichs wichtige Programmteile aufzeigen. Die Entwicklung der Benutzerrollen und Task Cases wird mittels Indexkarten dokumentiert. Ein weiteres modellorientiertes Artefakt zur Erfassung besonderer Aspekte der Arbeitsumgebung und des Anwendungskontexts eines Systems ist das Operational Model (Constantine & Lockwood 1999). Die wichtigsten Aspekte werden auf Indexkarten festgehalten, damit keine unnötige Dokumentation stattfindet und nicht zu viel Zeit benötigt wird.

Kaum erwähnt werden bei agilen Ansätzen Design-Prinzipien und Style Guides. Als Alternative zu umfassenden Fimen Style Guides schlagen wir die Entwicklung von light-weight Style Guides vor, die nur wenige Seiten umfassen. Auf Basis der herkömmlichen Style Guides wird eine sehr komprimierte Form verfasst. Damit Programmierer diese dann einfacher anwenden können, sollte der light-weight Style Guide darüber hinaus um UI- bzw. Interaction-Patterns (Borchers 2001; van Welie & Traetteberg 2000) ergänzt werden. Letztere sind wesentlich implementierungsorientierter und ermöglichen daher eine schnellere Umsetzung der Style Guide-Prinzipien. Angewendet nach dem AM-Prinzip „Apply Design Standards" und „Use Existing Ressources" (Ambler 2002) sind die light-weight Style Guides und UI-/Interaction-Patterns eine gute Basis für unseren AUCSL. Unabdingbar ist aber eine gute Qualifikation der Programmierer, d.h. sie sollten zumindest Erfahrung im Umgang mit Style Guides haben und die grundlegenden Gestaltungsrichtlinien der jeweiligen Entwicklungsplattform kennen (z.B. Java Design Guidelines, Sun Microsystems 2004).

Um die IRUUF abzuschließen, müssen schließlich noch Usability-Ziele aufgestellt werden. Dazu stehen die bereits ermittelten Artefakte, im Speziellen die Benutzerbedürfnisse (z.B. „Was wollen die Benutzer beim Einsatz der Software erreichen?", „Was motiviert sie diese Software zu benutzen?"), zur Verfügung. Die gesamte IRUUF sollte nur wenig mehr Zeit als die IRUF des XP benötigen. Wir denken, dass dies durch den Einsatz agiler UE-Techniken, die Einsparung von Dokumentation sowie im obigen Sinne angepasster Style Guides ergänzt um UI-Patterns und gut ausgebildete Entwickler möglich ist.

3.2 Initial Conceptual Phase

In die zweite Phase (Initial Conceptual Phase) unseres Prozesses fließen als Grundlage die in der IRUUF entwickelten Artefakte ein. Wir haben in dieser Phase eine Trennung der Entwicklung des UI und der Systemarchitektur vorgenommen. Dies ist vor allem bei größeren Projekten sinnvoll. Voraussetzung ist allerdings, dass sich die Programmierung einfach in Funktionalität und Grafik aufteilen lässt. Eine solche Trennung kann mit Hilfe von Software Patterns (Gamma et al. 1997) erreicht werden. Ist diese Separation nicht möglich, sollte das Entwicklerteam bei der Implementierung eng zusammenarbeiten.

Es steht fest, dass der Einfluss der Benutzeroberfläche auf die Architektur mit steigendem Maß an Interaktivität zunimmt. Die Konzeption vieler UI-unabhängiger Komponenten der Systemarchitektur sollte nicht auf die Fertigstellung des UI warten müssen, sondern kann parallel zur Verfeinerung der Szenarien stattfinden. Welche der Komponenten unabhängig sind, kann bei der Analyse der Szenarien festgestellt werden. Um im weiteren Verlauf UI und Architektur parallel entwickeln zu können, müssen diese aufeinander abgestimmt werden. Dazu muss eine minimalistische UI-Spezifikation erstellt werden (Constantine & Lockwood 2002), die den Entwicklern der Systemarchitektur als Vorlage für die Schnittstellen der abhängigen Komponenten dient. Sie beinhaltet die Organisation und Anordnung aller für die Aufgaben der Benutzer erforderlichen Elemente, ein Schema für die Navigation sowie ein Interaktionsschema für die konsistente Gestaltung und Verwendung der Software.

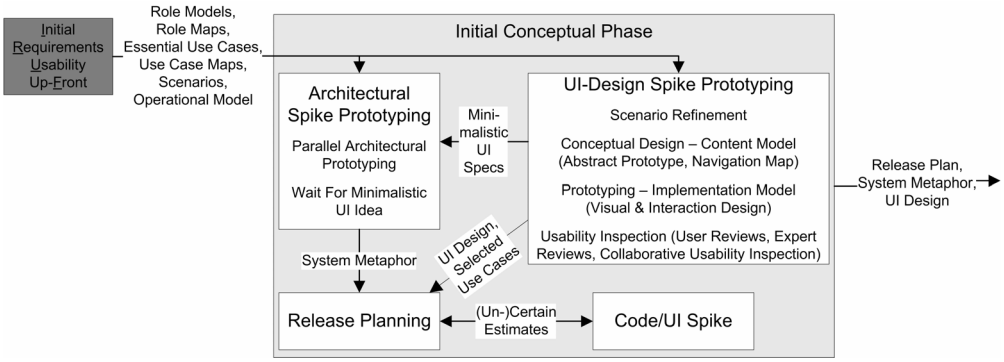

Abbildung 3: Initial Conceptual Phase des AUCSL

Nachdem die minimalistische Spezifikation entwickelt ist, beginnt das Screen Design Prototyping. Die in der IRUUF erzeugten Task Cases und Szenarien sind die Basis für den Entwurf von Low-Fidelity und High-Fidelity Prototypen. Welche Arten von Prototypen dabei eingesetzt werden, muss je nach Situation abgewogen werden. Papierprototypen können für das schnelle Testen erster Ideen genutzt werden, während beispielsweise Macromedia Flash basierte Prototypen ideal für Präsentationen vor Publikum und die Simulation der Interaktion sind. Die Erstellung von realistischen Prototypen ist aufwändiger, diese können jedoch in der Construction Phase eventuell weiterverwendet werden.

Erste Evaluationen werden schon in der konzeptionellen Phase durchgeführt. Agile Usability Techniken, die eingesetzt werden können, ohne viel Aufwand zu betreiben, sind zum Beispiel User und Expert Reviews sowie Collaborative Usability Inspection (Constantine und Lockwood 1999). Wichtig ist ein regelmäßiger Austausch von Benutzern, damit diese sich nicht nach und nach an das Projekt gewöhnen und damit zu vorbelasteten Testpersonen werden. Alle hier angesprochenen Techniken liegen im Ermessen des UE-Expertenteams und sind durch den vorgegebenen Zeitplan eingeschränkt. Durch so genannte Spike Solutions werden Antworten zu technischen Problemstellungen oder Designfragen erörtert. Spike Solutions bestehen aus kleinen Programmteilen, die versuchen bestimmte technisch relevante Aspekte oder Designideen umzusetzen (Proof of Concept). Dadurch wird das Risiko falscher Designentscheidungen minimiert.

3.3 Construction Phase

Mit der Construction Phase beginnt die eigentliche Programmierung der Software. In unserem Prozess verfolgen wir die Idee der iterativen, inkrementellen Implementierung des XP.

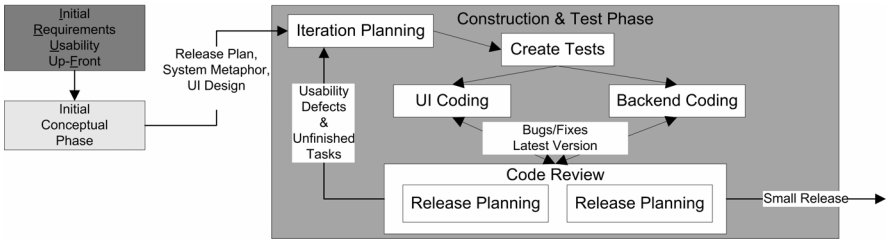

Abbildung 4: Construction und Test Phase des AUCSL

In diese Phase fließen der Release Plan, die Systemmetapher und das UI-Design aus der Initial Conceptual Phase ein. Bevor die Programmierung beginnen kann, wird ein Iteration Planning wie im XP durchgeführt. Auch die Tests und Programmierung werden analog zum XP durchgeführt (z.B. Techniken wie Pair Programming).

Neben den funktionalen Unit-/Acceptance-Tests, die später automatisiert ablaufen können, werden hier auch die Usability-Testfälle (z.B. mögliche Testaufgaben) im Vorfeld überlegt. Dies stellt sicher, dass bei der Implementierung auch die entsprechenden Usability Goals nicht aus dem Blickfeld geraten. Diese starke Testorientierung ist unserer Meinung nach eine der entscheidenden Stärken von XP im Vergleich zu herkömmlichen SE-Prozessen.

Als nächstes findet die teilweise parallel ablaufende Implementierung des UI und der Systemarchitektur statt. UI-Komponenten, die für die Implementierung der Systemarchitektur schnell angeliefert werden müssen, können mit geringer Genauigkeit umgesetzt und in späteren Iterationen verfeinert werden. Agile Usability-Tests finden parallel zu den automatisierten Tests statt, müssen aber nur durchgeführt werden, wenn Usability-Experten es als notwendig ansehen. Um in den Iterationen Usability-Tests einzusparen und damit den Zeit- und Kostenaufwand zu reduzieren, ist es sinnvoll Usability-Experten in die Entwicklerteams zu integrieren.

Sind die ersten Programmteile fertig gestellt, kann ein Small Release herausgegeben werden. Dieses ist unter Umständen zunächst noch ein Prototyp, der in einer möglichst schnellen Evaluation getestet wird. Agile Methoden, die an dieser Stelle eingesetzt werden können, sind Expert Reviews, Walkthroughs und im Aufwand reduzierte Benutzertests (z.B. Verzicht auf UE Lab, einfache Protokollierungstechniken). Die Gebrauchstauglichkeitsprobleme (Usability Defects), die dabei ermittelt werden, können einfach auf Indexkarten (Defect Cards) festgehalten und den Szenarien zugeordnet werden. Auf Basis der entdeckten Usability Defects wird beim nächsten Iteration Planning entschieden, ob bei der weiteren Implementierung neue Funktionalität dazu kommt oder ob Usability Defects beseitigt werden sollen. Der

Usability-Experte hat dabei die entscheidende Rolle, den Kunden die Defekte zu kommunizieren und mit diesen abzuwägen, wie bei der weiteren Implementierung vorgegangen werden soll.

Die inkrementelle Vorgehensweise, d.h. ein stetiges Verändern des UI, wirkt sich unter Umständen negativ auf Lernförderlichkeit, Erwartungskonformität und Selbstbeschreibungsfähigkeit aus, ist mit UE jedoch grundsätzlich vereinbar. Steht einmal die minimalistische Architektur fest, so kann das UI inkrementell verfeinert werden. Ausgewählte Low-Fidelity Prototypen werden, unter Beibehaltung bereits evaluierter Inhalte, zu High-Fidelity Prototypen und Small Releases weiterentwickelt.

In der flexiblen Construction Phase ist es wie im XP möglich, auf Änderungswünsche des Kunden und neue Anforderungen einzugehen. Vom Standpunkt des UE aus müssen Refactoring-Aktivitäten jedoch hinsichtlich der Konsistenz mit dem entwickelten konzeptionellen UI-Architekturmodell (siehe 3.2 Initial Conceptual Phase) geprüft werden. Werden dabei Inkonsistenzen ausgemacht, muss nicht nur der Zeit- und Kostenaufwand geschätzt werden den, die Änderungen verursachen würden. Bei elementaren Änderungen kann es zudem notwendig sein, neue Anforderungen zu erheben und damit in die IRUUF zurückzuspringen. Die Aufwandsschätzung wird dann mit dem Kunden abgesprochen und dieser kann entscheiden, ob die gewünschten Änderungen in das Iteration Planning aufgenommen werden sollen.

3.4 Deployment und Production Phase

In Deployment und Production Phase ist ein Auftreten von unvorhergesehenen Problemen oder Usability Defects möglich. Es kann vorkommen, dass schwerwiegende Probleme oder komplett neuartige Funktionalität (Tests/Kundenfeedback) zu Änderungen am Konzept oder einer Ergänzung der Anforderungsermittlung führen. Daraus resultieren kostspielige Änderungen, die sich auf das ganze Projekt auswirken. Durch unsere Vorgehensweise soll genau das vermieden werden, indem schon in frühen Phasen das Risiko durch eine ausreichend genaue Anforderungsermittlung und frühes Prototyping minimiert wird.

4 Resümee und Ausblick

Mit der von uns vorgestellten Vorgehensweise erweitern wir XP um AM- und UE-Praktiken mit dem Ziel gebrauchstaugliche Software agil entwickeln zu können. Als Fazit schließen wir aus unseren theoretischen Überlegungen, dass es möglich ist UE in agile Vorgehensweisen, wie beispielsweise XP, zu integrieren. Wir haben dazu bewusst bereits agile Methoden und Techniken des UE vorgeschlagen und wo immer es uns möglich erschien, die erforderliche Dokumentation reduziert („agilisiert“). Im Vergleich zum ursprünglichen XP Lifecycle ist der hier vorgestellte sicher methodisch und auch die Zusammensetzung des Entwicklungsteams betreffend umfassender und daher wohl weniger agil. Dafür reduzieren wir aber aus dem Blickwinkel des UE das Risiko, keine gebrauchstaugliche Software zu entwickeln! Insgesamt erscheint es uns aber immer noch als gerechtfertigt, von einem agilen Lifecycle zu

sprechen, da wir die grundlegenden XP- bzw. AM-Praktiken beibehalten haben und bei der methodischen Anreicherung sehr auf agilisierte UE-Methoden Wert gelegt haben.

Literaturverzeichnis

AgileAlliance (2004): http://www.agilealliance.com/home (letzter Zugriff: 20.2.2004).

Ambler, W. Scott (2002): Agile Modeling. New York: John Wiley & Sons.

Beck, K. (1999): Extreme Programming Explained. Addison-Wesley.

Beyer H., Holtzblatt K. (1998): Contextual Design. Morgan Kaufmann.

Borchers, J. (2001): A Pattern Approach to Interaction Design. N. Y.: John Wiley & Sons.

Cockburn, A. (2001): Agile Software Development. Addison-Wesley.

Constantine, L. L. (2002): Process Agility and Software Usability: Toward Lightweight Usage-Centered Design. Information Age.

Constantine, L. L.; Lockwood, Lucy A. D. (1999): Software for Use: A Practical Guide to the Methods of Usage-Centered Design. Addison-Wesley Professional.

Dzida, W.; Freitag, R. (1998): Making Use of Scenarios for Validating Analysis and Design. In: IEEE Trans. Soft. 24 (12), S. 1182–1196.

EC Wise Inc. (2003): Integrating User Centered Design in Agile Development processes. http://www.ecwise.com/whitepapers/ (letzter Zugriff: 07.11.2003).

Gamma, E.; Helm, R.; Johnson, R. E. (1997): Design Patterns. Addison-Wesley.

Mayhew, D. J. (1999): The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design. San Francisco, CA: Morgan Kaufmann Publishers.

Preece, J.; Rogers, Y.; Sharp, H. (2002): Interaction Design. N.Y.: John Wiley & Sons.

Rosson, M. B.; Carroll, J. M. (2001): Scenario-Based Usability Engineering. Redwood City, CA: Morgan Kaufmann Publishers.

Sun Microsystems (2004): Look and Feel Design Guidelines. 2nd Edition. http://java.sun.com/products/jlf/ed2/book/ (letzter Zugriff: 20.2.2004).

van Welie, M.; Traetteberg H. (2000): Interaction Patterns in User Interfaces. 7th Pattern Languages of Programs Conference, 13.–16. August 2000, Monticello Illinois, USA.

Kontaktinformationen

Prof. Dr. Harald Reiterer / Fredrik Gundelsweiler / Thomas Memmel
Universität Konstanz: FB Informatik und Informationswissenschaft
Postfach D73; 78457Konstanz;
E-Mail: harald.reiterer@uni-konstanz.de; fredrik.gundelsweiler@uni-konstanz.de;
thomas.memmel@uni-konstanz.de; http://hci.uni-konstanz.de/