XP + UE → XU Praktische Erfahrungen mit eXtreme Usability

"I've consciously decided to give up the ability to predict the future."

(Kent Beck, 2002)

Andreas Holzinger · Wolfgang Slany

Die Erfolgsfaktoren von Extreme Programming (XP) basieren auf einer optimalen Kommunikation im Programmierteam, in der Einfachheit der Vorgehensweise und vor allem auf den häufigen Releases und den sich damit ergebenden Reaktionsmöglichkeiten auf sich verändernde Anforderungen.

Der Kunde ist dabei in den Entwicklungsprozess vollständig eingebunden und gibt ständiges Feedback. Im Usability Engineering (UE) ist der „Kunde" der End-Benutzer und die Vorgangsweise ist ähnlich, wobei sich das traditionelle Usability Engineering (UE) auf die Gestaltung von Benutzerschnittstellen konzentriert und auf einem spiralförmigen vierphasigen (Analyse, Entwurf, Entwicklung, Test) und dreistufigen (Papiermodell, Prototyp, Endprodukt) Vorgehensmodell beruht. Bei XP sind diese Phasen im Vergleich zum UE extrem verkürzt. Werden die Erfolgsfaktoren von Extreme Programming und die Erfolgsfaktoren des Usability Engineering verbunden, erhalten wir einen neuen Ansatz: Extreme Usability (XU). Die Autoren machten sehr gute Erfahrungen mit der Anwendung von XU in der Praxis kleinerer Projekte im medizinischen Umfeld und insbesondere in der Ausbildung von Software-Ingenieuren und berichten in diesem Artikel über ihre Erfahrungen.

Extreme Programming

Extreme Programming (XP) ist ein (umstrittener) Softwareentwicklungsprozess [13], der die häufigsten Probleme in der Softwareentwicklung in den Griff bekommen will [1, 2, 9]. Das „extreme" im Namen bezieht sich darauf, dass die „best practices" der Softwareentwicklung in „extremer Weise" so kombiniert werden, dass ihre individuellen Schwächen kompensiert werden (sollen).

Kent Beck, der „Vater" von XP, definiert die Praktiken dieser Methode folgendermaßen (siehe auch Abb. 1):

Andauernde Code Reviews durch Programmieren in Paaren, die halbtäglich oder öfter wechseln, sodass nach kurzer Zeit jeder mit jedem zusammengearbeitet hat; ständiges Testen, auch *täglich* durch Kunden, ermöglicht durch permanente Präsenz zumindest eines Kunden; klarstes, einfachst mögliches Design (YAGNI-Prinzip: „you aren't gonna need it"); monatliche Plananpassungen mit Kunden (dies sind allerdings die einzigen „*Sitz*ungen" des gesamten Teams; sonst gibt es nur tägliche 5-minütige *Stand-Up*-Meetings); fortlaufende Integration mit extrem kurzen Iterationszeiten: Sekunden, Minuten und Stunden statt Wochen, Monaten und Jahren; Releases im Dreimonatsrhythmus oder kürzer, dadurch Fokussierung auf Wesentliches und Machbares; bewusste Verwendung einfacher Metaphern, die sowohl Kunden als auch Entwicklern eine effiziente Kommunikation ermöglicht; 40-Stunden-Woche für

DOI 10.1007/s00287-006-0060-5

Andreas Holzinger
Institut für med. Informatik,
Med. Universität Graz,
E-Mail: andreas.holzinger@meduni-graz.at
und
Institut für Software Technik,
Technische Universität Wien,
E-Mail: holzinger@ifs.tuwien.ac.at

Wolfgang Slany
Institut für Softwaretechnologie (IST)
Technische Universität Graz,
E-mail: wsi@ist.tugraz.at

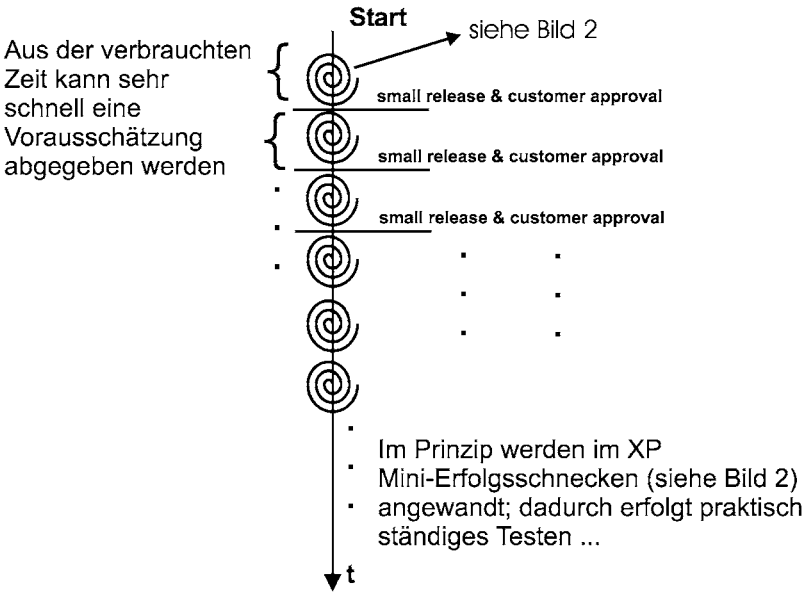

Abb. 1 Ein Erfolgsprinzip von XP, vgl. mit Abb. 2

nachhaltige Entwicklung; keine Spezialisierung der Entwickler: Der Code gehört allen, wird von allen verstanden und *von allen* refactored, ermöglicht durch deklarative, automatische und vollständige Test-Abdeckung (Test-first-Praktik).

Charakteristisch sind die zeitlich extrem verkürzten Planungszyklen und die Konzentration auf das, was machbar und für den Kunden am wichtigsten ist. So genannte „User Stories" (ähnlich den etwas größeren „Szenarien" im Usability Engineering) treiben den Entwicklungsprozess. Usability-Aspekte werden dabei gleich wie andere Features behandelt. Der Kunde reiht *in Diskussion mit den Entwicklern* die User Stories nach ihrer absoluten Wichtigkeit für den wirtschaftlichen Erfolg des Projekts, hat allerdings *zu 100 %* das letzte Wort dabei. Durch die Diskussion mit den Entwicklern wird sichergestellt, dass die Kreativität der Entwickler trotzdem dem Projekt zugute kommen kann.

Jeder User Story werden mehrere Engineering Tasks zugeordnet. Engineering Tasks ohne Bedarf für die minimale Realisierung einer User Story werden nicht implementiert: „simplicity – the art of maximizing the amount of work *not* done – is essential"[1]. Eine User Story beschreibt ein aus Sicht des Kunden nicht weiter aufteilbares Feature, das – falls es gerade implementiert wird – von maximalem Nutzen für den End-Benutzer ist. Das inkludiert nun zwangsläufig Usability-Aspekte, die natürlich stets in Zusammenhang zur sonstigen Funktionalität stehen müssen.

Zu jeder Story wird ein „Testfall" entwickelt, der die Funktionalität der Story feststellt, wobei hier auch wieder Usability-Aspekte z. B. empirisch, zumindest aber möglichst objektiv nachvollziehbar, getestet werden. Die Entwicklung einer Story wird abgebrochen, wenn alle Testfälle zur Erfüllung aller Stories erfüllt sind, die bisher implementiert wurden. Dies bedeutet, dass Engineering Tasks, die zu (noch) nicht implementierten User Stories gehören und die für bisher implementierte User Stories nicht *absolut* notwendig sind, *nicht* implementiert werden, bewusst die Gefahr in Kauf nehmend, dass dies später zu großen Änderungen führen könnte (YAGNI-Prinzip: „you aren't gonna need it").

Während des Projekts auftretende Änderungswünsche der Kunden werden als Vorteil für das Projekt willkommen geheißen und als solche (zumindest theoretisch problemlos) vom restlichen Team akzeptiert, unter Bewusstmachung der daraus resultierenden Kosten bzw. der Notwendigkeit, bereits eingeplante User Stories auf spätere Iterationen oder gar Releases verschieben zu müssen. Durch die extrem verkürzten Release-Zyklen wird trotzdem sichergestellt, dass bei jeder Release ein in sich abgeschlossenes Produkt fertig gestellt, dokumen-

[1] The agile manifesto, http://agilemanifesto.org/

tiert und getestet ist, das den unter den gegebenen Ressourcen maximalen Endnutzen besitzt. Dadurch werden Release-Zeitpunkte *und* tatsächliche Entwicklungskosten planbar.

Durch die Anwendung der Praktiken des XP kann auf Personalwechsel im Vergleich zu anderen Entwicklungsmethoden problemloser reagiert werden: Jedes Teammitglied ist austauschbar, lernt allerdings auch intensiv von allen anderen im Team, was vom Standpunkt der Organisation zu dem bestmöglichen Wissensmanagement im Projekt führt. Auf Artefakte, die für den End-Benutzer gar nicht von unmittelbarer Relevanz sind, wird gänzlich verzichtet (dies schont auch die Wälder).

Aufgrund der besseren Kommunikation wird auf nicht unmittelbare oder nicht automatisch testbare Kommunikation verzichtet, wodurch wegen Ersterem die Teamgröße auf (maximal) 12 Personen begrenzt ist.

XP kann daher als pragmatischste der modernen Softwareentwicklungsmethoden bezeichnet werden, mit dem gleichzeitig geringsten Frustrationsfaktor für die Entwickler, abseits des chaotischen „cowboy development"-Stils, der allerdings auch signifikant seltener als XP zum nachhaltigen Erfolg in kommerziellen Projekten führt. Ein bekanntes Beispiel: XP wurde für die Lohnverrechnung bei Daimler Chrysler mit dem Ziel entwickelt, qualitativ hochwertige Software unter realistischer Einschätzung der vorhandenen Ressourcen zu erstellen. Allerdings bemerkt selbst Kent Beck, dass die Praktiken des XP – Testen ausgenommen – für sich genommen gar nicht sonderlich wirksam sind. Erst das Zusammenspiel (der Mix) macht XP aus. Und eben genau dieses Zusammenspiel zu erreichen, kann sich in der Praxis als durchaus schwierig erweisen.

Das beginnt bereits vor dem Projektstart. Der typische Budgetgedanke – die Lieferung eines Produkts mit einem festgelegten Umfang zu einem bestimmten Datum für einen fixen Preis – kollidiert unweigerlich mit dem stark iterativen Vorgehen. Hinter diesen Vorgaben stecken durchaus berechtigte Anforderungen aus der Unternehmensführung und dem Controlling auf Seiten des Kunden, die den Handlungsspielraum seines Projektleiters beschränken. Hier ist schon im Vorfeld kommunikations- und zeitaufwändige Überzeugungsarbeit zu leisten, um ein Projekt auf Basis dieses Entwicklungsprozesses überhaupt an den Start zu bringen.

Als weiteres Beispiel sei die Präsenz des Kunden vor Ort (*On-Site Customer*) genannt. Sie soll sicherstellen, dass der Auftraggeber genau das bekommt, was er möchte (wenn der Kunde weiß, was er will). Sofern der Kunde dies akzeptiert, setzt das auch voraus, dass der Ansprechpartner des Kunden tatsächlich über die Kompetenz (nämlich Fachwissen *und* Befugnisse) verfügt, um klare Entscheidungen treffen zu können. Oftmals ist aber genau das nicht oder nur sehr schwer realisierbar. Daher ist es allzu oft bei Projekten mit wenig Rückhalt im Management ausgesprochen schwierig, zu klaren Aussagen – etwa darüber, welche User Stories im nächsten Release umgesetzt werden sollen – zu gelangen, da sich niemand wirklich dafür zuständig fühlt und/oder sich diesbezüglich exponieren möchte.

Nicht nur einmal bekommt man dann zu hören: „So habe ich mir das aber nicht vorgestellt!". Das ist eine typische Reaktion bei der Vorstellung des Produkts – allerdings *nicht* vom Ansprechpartner auf Seiten des Kunden, der ja in die Entwicklung mit einbezogen war, sondern von seinen Vorgesetzten, vom Management, das häufig auch über die Freigabe der finanziellen Mittel entscheidet.

Das lässt naturgemäß die Sinnhaftigkeit einer schriftlichen und vom Kunden unterzeichneten Anforderungsdefinition, auf die in der „reinen Lehre" des XP verzichtet wird, in einem anderen Licht erscheinen.

In weiten Bereichen suggeriert XP, es hätte für die tatsächlichen Probleme eine – dazu noch einfache – Lösung. Sowohl aus betriebswirtschaftlicher als auch aus rechtlicher und technischer Sicht ist das oft *zu einfach* und insbesondere bei Projekten mit externen Auftraggebern sehr optimistisch und idealisiert gesehen. Dessen sollte man sich bei der Adaptierung leichtgewichtiger Entwicklungsprozesse an die eigenen Gegebenheiten bewusst sein. XP propagiert hauptsächlich *Werte* von durchaus allgemeiner Gültigkeit (im Gegensatz zu *Regeln*), die in Bezug auf ihre Anwendung einen sehr breiten Interpretationsspielraum zulassen. Der typische XP-Projektleiter benötigt daher für das Extreme Programming selbst einiges an Erfahrung und weiteren Kenntnissen und Fähigkeiten. Die Gefahr besteht darin, vorzugeben XP zu betreiben und damit lediglich den Verzicht auf Anforderungsanalyse, Entwurf und/oder Dokumentation zu meinen!

Usability Engineering (UE)

Es gibt allerdings einen Faktor, der für Software-Systeme *aller Art* immer wichtiger wird: gute Usability [3, 10, 14, 15]. Um gute Usability zu erreichen, kann das Usability Engineering Vorgehensmodell (siehe Abb. 2) angewandt werden.

Dabei wird das Ergebnis von Anforderungsanalyse und Grundkonzept in Form von Skizzen auf Papier gebracht (Paper Mock-ups, siehe Abb. 3). Mit diesen Papiermodellen können nun sofort Usability-Tests mit End-Benutzern durchgeführt werden [5].

Dabei wird meistens eine konkrete Aufgabe gestellt und z. B. untersucht, wie lange die End-Benutzer zur Bearbeitung der Aufgabe benötigen (Time to perform task) und welche Schwierigkeiten sie bei den Interaktionen haben (cognitive effort). Untersuchungen haben gezeigt, dass End-Benutzer zu solchen Papier/Bleistift-Modellen mehr Aussagen machen, als wenn sie (gleich) mit einem Modell am Computer oder gar mit einem laufenden System konfrontiert werden [10].

Die Ergebnisse aus diesen frühen Usability-Tests, bei denen bereits rund 80% aller Schwierigkeiten entdeckt werden, fließen sofort in die Entwicklung ein [12].

Bereits von Anfang an (Analyse, Design, Grobkonzept, Prototyping, Usability-Tests) herrscht hier eine enge Kommunikation zwischen einem oder zwei Softwareentwickler(n) und dem End-Benutzer (siehe Abb. 3 und 4).

Die Ergebnisse, die traditionell im sequentiellen Softwareentwicklungsmodell „nur" als Dokumentation vorhanden sind, werden hier tatsächlich ausgeführt und getestet. Dieses frühe Testen führt zu einer hohen Wahrscheinlichkeit, Fehler früh zu erkennen, was die Kosten für spätere Fehlerbeseitigung enorm senkt [11]. Das zyklische, iterative Vorgehen bringt den Softwareentwicklern durch das Feedback der End-Benutzer rasch überprüfbare Ergebnisse, mehr Motivation und schließlich eine

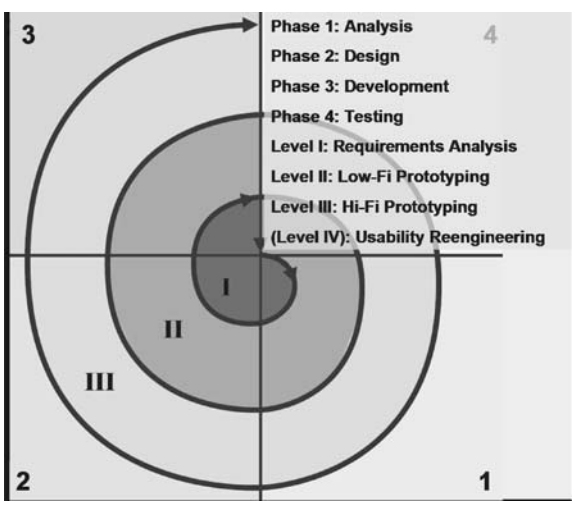

Abb. 2 Die Erfolgsschnecke [6]

Abb. 3 Usability Tests (hier: Thinking Aloud mit Video-Analyse) mit Paper Mock-Ups sind sehr erfolgreich zum Testen von Interface Entwürfen

Abb. 4 Unmittelbar nach Durchführung einer Real-Life-Usability-Studie (links) werden die Ergebnisse von den Software-Ingenieuren analysiert und entsprechend umgesetzt (rechts)

erhebliche Qualitätsverbesserung im Softwareentwicklungsprozess [4, 5]. Die Produktivität des Teams wird dadurch gesteigert, vor allem aber reduziert sich das Risiko des Scheiterns eines Projekts, da experimentell bestätigte Ergebnisse auch ein Maß an Sicherheit (gegenüber dem Auftraggeber) bieten.

Die Gefahr bei dieser Methode liegt allerdings in der Verzettelung beim Anwender durch eine zu große Detailierungssucht. Rapid Prototyping erfordert daher den Mut, unvollkommene Prototypen zu bauen. Andererseits besteht die Gefahr, dass trivial erscheinende, aber schwierig zu implementierende Details weggelassen werden, die dann später die Entwicklungskosten wieder ansteigen lassen. Ein Nachteil ist auch, dass es bei dieser Methode sehr schwer ist, ein Pflichtenheft zu führen, denn dieses müsste ja nach jedem Zyklus angepasst werden, was für eine Vertragsgestaltung wiederum sehr schwierig sein kann.

Extreme Usability (XU)

Der Gefahr der Verzettelung in Details und der Detaillierungssucht der Kunden (aber besonders auch der Entwickler) beim UE wird durch die 12 Praktiken des XP bewusst Einhalt geboten:

1) realistische Abschätzungen dessen, was bis zur nächsten Release möglich ist, und Fokussierung auf das momentan am wichtigsten Erscheinende, mit häufiger Änderungsmöglichkeit in dieser Prioritätenliste;

2) Verzicht darauf, längerfristig zu planen;

3) Lizenz zum Verzicht des Nachdenkens über das, was später kommen könnte und dafür bereits jetzt sinnvoll wäre, mit Bereitschaft, später umfangreiche Anpassungen und Änderungen vorzunehmen.

Die zentrale Frage lautet nun: Wo kann nun das Usability Engineering in den Extreme Programming Zyklus eingefügt werden? Wo also kann man die beiden Methoden „verheiraten“? Im XP gibt es zu jedem Zeitpunkt einer Iteration eine *wachsende* Menge von fertigen User Stories und eine *abnehmende* Menge von unfertigen User Stories. Da fortlaufend integriert wird, gibt es zu jedem Zeitpunkt ein Softwaresystem von *maximalem* Kundennutzen, das vom Kunden (= hoffentlich End-Benutzer) auch bereits funktional getestet wurde. Es ist daher zu jedem Zeitpunkt möglich *und* auch sinnvoll, die Usability des bereits implementierten Teilprodukts von unabhängigen Dritten testen zu lassen. Solcherart gewonnene Einsichten können entweder in einer nächsten Iteration (die ja jeweils nur ein bis vier Wochen dauert) umgesetzt werden, oder, falls es sehr dringend ist, weil zum Beispiel eine Deadline (wie fast immer) vorliegt, gleich innerhalb der momentanen Iteration als neue User Story eingeplant werden, was aber natürlich bewirkt, dass andere eingeplante User Stories entsprechend auf einen späteren Zeitpunkt verschoben werden müssen.

Weiter ist es möglich, *noch nicht* implementierte User Stories mittels Paper Prototyping, das zusammen mit dem Kunden (End-Benutzer) vor Ort zusammengestellt wird, im Voraus von unabhängigen Dritten testen zu lassen. In diesem Fall ist die Auswirkung auf die zu implementierende User Story unmittelbarer und daher im Idealfall vorzuziehen. In der Praxis lassen sich allerdings Usability-Tests einzeln für jede User Story (denn weiter wird ja wegen der Simple Design Praktik nicht im Voraus detailliert nachgedacht) nur selten realisieren, man nimmt daher mit dem Testen der Usability durch unabhängige Dritte einer Mischung aus einem bereits implementierten Teilsystem und einem bereits absehbaren, aber noch nicht implementierten Paper Prototypen vorlieb. Durch die hohe Frequenz der Iterationen hat man allerdings immer noch eine gute Balance zwischen Usability-Tests, die eine reale Auswirkung auf das Endprodukt haben, den damit verbundenen Kosten und der Vermeidung von Usability Tests von Systemteilen, die letztendlich aus den verschiedensten Gründen gar nicht implementiert werden.

Extreme Usability sieht daher so aus, dass alle „best practices“ des UE *während* einer Iteration im XP-Prozess eingehalten werden können, solange dies gewünscht ist, mit einer Einschränkung der Usability Aspekte auf die vorliegende Iteration. Der Vorteil ist, dass beim XP-Prozess die Anpassung und graduelle Verfeinerung bis zum Ende des Projekts explizit in den Prozess eingebaut ist, was dem UE wiederum sehr entgegenkommt. UE kann daher die XP-Entwicklungsmethode definitiv befruchten, indem der Fokus auf die wichtigen Aspekte der Usability gelenkt werden und diese dem Kunden durch das gesamte Entwicklungsteam permanent (durch tägliches Nachfragen, Diskutieren und Testen) bewusst gemacht werden; auch die Sinne der Entwickler werden auf die wichtigen Usability-Aspekte fokussiert, wenn zumindest ein

Entwickler im Team Vorwissen über UE besitzt und dieses an die anderen Entwickler und den Kunden durch Paarprogrammierung samt vollständiger und häufiger Durchmischung der Paare sowie der XP-Praktik des Kunden-vor-Ort XP weitergibt. Durch die Kunde-vor-Ort-Praktik gibt es auch ein dauerndes Testen des Systems durch einen Endbenutzer, sodass Usability-Probleme wesentlich rascher als in herkömmlichen Entwicklungsmethoden erkannt und weiter durch die im XP betonte Flexibilität (Praktik der short-releases, des häufigen Planungsspiels mit Kurs-Korrekturmöglichkeiten) ebenso rasch gelöst werden.

UE-Erfahrung für alle Entwickler ist natürlich in jedem Projekt von Vorteil. Extreme Usability, die Symbiose aus XP und UE, könnte so zu etwas optimaleren als den beiden erfolgreichen Methoden selbst kombiniert werden.

Praktische Erfahrungen

Die Autoren sammelten in zwei Lehrveranstaltungen im Sommersemester 2005 Erfahrungen in der Ausbildung von Software-Ingenieuren: in einer Großlehrveranstaltung mit 150 Studierenden und in einer praktisch orientierten Speziallehrveranstaltung mit 30 Studierenden. Die Studierenden hatten dabei bereits (theoretische) Vorkenntnisse, sowohl aus XP als auch aus UE. In der Großlehrveranstaltung wurden die Studierenden in Gruppen (Teams) von maximal 10 Studierenden eingeteilt, in der Speziallehrveranstaltung in Gruppen von maximal 4 Studierenden. Die 10-Personen-Gruppen enthielten jeweils einen End-Benutzer (Kunden), einen Manager, einen Coach/Entwickler und „normale“ Programmierer. Während der Lehrveranstaltung mussten die Gruppen jeweils ein vorher ausgewähltes Projekt bearbeiten, das stets einen starken User-Interface-Aspekt enthielt. Die Gruppen arbeiteten in 8 Einheiten zu jeweils 8 Stunden. Als Usability-Tester standen den Teams externe Dritte zur Verfügung. Hier bestand unserer Meinung nach die Schwachstelle unseres Experiments: Gewöhnlicherweise haben die Kunden (End-Benutzer) nämlich weder Informatikwissen, noch sind diese überhaupt technisch bewandert oder interessiert. Als externe Dritte wurden allerdings andere Technik-Studierende eingesetzt. Daher ist unsere nächste Idee: das Einbringen von „echten“ Kunden bzw. „echten“ End-Benutzern. Eine Gruppe in der kleineren Lehrveranstaltung hat das auf Eigeninitiative hin gemacht und testete ihr medizinisches Projekt im Real-Life-Umfeld eines Arztes. Dabei wurden Usability-Tests mit der Methode „Thinking aloud“ durchgeführt [7]. Das ist eine Methode, bei der die End-Benutzer aufgefordert werden, alles auszusprechen, was ihnen während der Bearbeitung der Tasks durch den Kopf geht. Die Studierenden erhielten so eine profunde Einsicht in das Benutzerverhalten und konnten ihr Projekt entsprechend verbessern (siehe Abb. 3).

Während der planning games wurden stets Usability-Aspekte untersucht. Es wurde konsistent die Methode „Thinking aloud“ mit Videoanalyse an paper mock-ups angewandt, um die Funktionalitäten der User Stories ausgiebig zu testen.

Am Ende beider Lehrveranstaltungen wurden in der größeren Lehrveranstaltung eine trade show und in der kleineren Lehrveranstaltung eine Mini-Konferenz durchgeführt, bei der die Studierenden ihre Projekte präsentierten und mit ihren Kolleginnen und Kollegen diskutierten.

Sehr gute Erfahrungen wurden in mehreren Kleinprojekten am Klinikum Graz gemacht. Dort gibt es eine Fülle von herausfordernden Aufgabenstellungen. Beispielsweise muss die interaktive ärztliche Dokumentation in einer stark frequentierten Ambulanz mit einem Minimum an Interaktivität auskommen. Das ärztliche Handeln muss durch den Einsatz des Computers bestmöglich unterstützt werden. Es ist unumgänglich, die Benutzeroberflächen so einfach und so intuitiv wie möglich zu gestalten. Dazu sind aber das genaue Umfeld, der Ablauf der Workflows und die genaue Kenntnis der End-Benutzer für die Software-Ingenieure unumgänglich. Die Möglichkeit, von den End-Benutzern zu lernen und Einblick in deren Probleme zu erhalten – bei gleichzeitiger sofortiger Umsetzbarkeit der Ergebnisse – wurde von den Software-Ingenieuren (siehe Abb. 4) enorm gut angenommen [8].

Zusammenfassung

Immer noch werden weltweit Usability-Aspekte entweder zu spät oder gar nicht in die Ausbildung von Software-Ingenieuren eingebracht. Lehrveranstaltungen, die sich auf praktische Anwendung von Usability Engineering konzentrieren, leisten daher einen wichtigen Beitrag zur Qualitätsverbesserung im Software Engineering.

Dabei kommt es darauf an, realistische, praxisorientierte Projekte durchzuführen (Research Based

Teaching) unter Einbeziehung von „echten“ End-Benutzern, möglichst im Real-Life. Es ist noch viel Arbeit notwendig, um die Lücke zwischen Theorie und Praxis zu schließen und um die „awareness“ der Wichtigkeit von Usability-Aspekten in die Ausbildung von Software-Ingenieuren zu integrieren. Jedenfalls ist die Kombination von Extreme Programming (XP) und Usability Engineering (UE) zur Methode Extreme Usability (XU) ein sehr viel versprechender Ansatz, um die Ausbildung in diesem Bereich zu verbessern. Im Sinne von: together today for a better software engineering of tomorrow!

Literatur

1. Beck, K.: Extreme Programming: Die revolutionäre Methode für Softwareentwicklung in kleinen Teams. München et al.: Addison-Wesley 2000
2. Cockburn, A.: Agile Software Development. Boston (MA): Addison-Wesley 2002
3. Eberleh, E., Oberquelle, H., Oppermann, R.: Einführung in die Software-Ergonomie. Berlin: deGruyter 1994
4. Holzinger, A.: Experiences with User Centered Development (UCD) for the Front End of the Virtual Medical Campus Graz. In: Jacko, J.A., Stephanidis, C. (Hrsg.) Human-Computer Interaction, Theory and Practice. Mahwah (NJ): Lawrence Erlbaum, S. 123–127, 2003
5. Holzinger, A.: Application of Rapid Prototyping to the User Interface Development for a Virtual Medical Campus. IEEE Softw. 21(1), 92–99 (2004)
6. Holzinger, A.: Schön, Einfach, Schlicht = Erfolgreich. Grundregeln des Web-Design & die Erfolgsschnecke. CHIP Professional 1(5), 10–17 (2004)
7. Holzinger, A.: Usability Engineering for Software Developers. Commun. ACM 48(1), 71–74 (2005)
8. Holzinger, A., Leitner, H.: Lessons from Real-Life Usability Engineering in Hospital: From Software Usability to Total Workplace Usability. In: Holzinger, A., Weidmann, K.-H. (Hrsg.) Empowering Software Quality: How can Usability Engineering reach these goals? Vienna: Austrian Computer Society 2005
9. Hruschka, P.: Agility. (Rück-)Besinnung auf Grundwerte in der Softwareentwicklung. Informatik-Spektrum 26(6), 397–401 (2003)
10. Nielsen, J.: Usability Engineering. San Francisco: Morgan Kaufmann 1993
11. Nielsen, J.: Using discount usability engineering to penetrate the intimidation barrier. In: Bias, R.G., Mayhew, D.J. (Hrsg.) Cost Justifiying Usability. Boston (MA): Academic Press 1994
12. Nielsen, J.: Usability Metrics: Tracking Interface Improvements. IEEE Softw. 13(6), 12–13 (1996)
13. Reißing, R.: Extremes Programmieren. Informatik-Spektrum 23(2), 118–121 (2000)
14. Shackel, B.: The Concept of Usability. In: Bennett, J., Case, D.J.S., Smith, M. (Hrsg.) Visual Display Terminals: Usability Issues and Health Concerns. Englewood Cliffs (NJ): Prentice Hall 1984
15. Stary, C.: Interaktive Systeme: Software Entwicklung und Software Ergonomie. Wiesbaden: Vieweg 1994