Ulrich Hasenkamp

Matthias Goeken

Alexander Schwartz

Extreme Programming als Vorgehensmodell für die Entwicklung von Groupware auf der Plattform Lotus Notes und Pavone Espresso

Fachbericht Nr. 01/01

Herausgeber:
Prof. Dr. Paul Alpar
Prof. Dr. Ulrich Hasenkamp
Philipps-Universität Marburg
Institut für Wirtschaftsinformatik
35032 Marburg
Telefon (06421) 282-38 94
E-Mail: {alpar | hasenkamp}@wiwi.uni-marburg.de

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Die Zeitschrift WIRTSCHAFTSINFORMATIK befasst sich mit Anwendungssystemen in Wirtschaft und Verwaltung, also mit Konzepten zur Planung, Entwicklung, Einführung, zum Einsatz und zur Wartung soziotechnischer Systeme. Diese Themen werden von Autoren und Herausgebern auch in Forschung und Lehre behandelt. Informations- und Kommunikationssysteme bilden aber auch eine wesentliche Grundlage für die Entstehung der einzelnen Ausgaben der Zeitschrift. So werden seit vielen Jahren die Aufgaben von Herausgebern und Redaktion durch mehrere Lotus-Notes-Datenbanken unterstützt. Mit einer Änderung der Aufgabenverteilung unter den Herausgebern und dem Wechsel einiger Mitarbeiter entstand Bedarf zur Anpassung der existierenden Applikationen. Deshalb wurde die Entscheidung getroffen, die Prozesse zu dokumentieren und zu analysieren sowie alle existierenden Applikationen zu *einer* neuen zu integrieren.

1.2 Auswahl eines Vorgehensmodells

Bei der Systementwicklung kommen zwei grundsätzlich verschiedene Ansätze in Frage: Phasenmodelle und evolutionäre Modelle. Phasenmodelle teilen die Softwareentwicklung in verschiedene Entwicklungsphasen auf, die nacheinander abgearbeitet werden.[1] Der Vorphase folgen Analyse, Entwurf und Realisierung. In der Einführung wird den Anwendern das Programm näher gebracht und der Produktionsbetrieb aufgenommen. Im anschließenden Systembetrieb werden die Entwicklungstätigkeiten heruntergefahren, es kommt nur noch zu Systemwartung und -pflege. Ein derartiges Vorgehen wird zunehmend kritisiert, weil es den Erfahrungen und Erwartungen der Anwender, dass immer wieder neue Anforderungen hinzukommen, widerspricht und wird daher für die Entwicklung sozial eingebetteter Systeme als ungeeignet angesehen. Empfohlen wird statt dessen eine „evolutionäre“ Vorgehensweise.[2] Da es sich bei dem betrachteten Vorhaben um ein eher kleines Softwareprojekt handelt, wurde auch Extreme Programming[3] in die engere Wahl genommen.

---

1 Vgl. Stahlknecht, Hasenkamp 1999 /Einführung Wirtschaftsinformatik/, S. 232ff.
2 Vgl. Hesse, Dahme 1997 /Software-Entwicklung/.
3 Vgl. Beck 1999 /Extreme Programming Explained/.

2 Extreme Programming (XP) als Vorgehensmodell

Um zu klären, ob Extreme Programming für dieses Projekt geeignet ist, müssen zwei Bereiche untersucht werden: Zunächst ist die generelle Eignung von XP für Groupwareprojekte zu prüfen. Danach ist zu klären, ob die technische Plattform Lotus Notes in Kombination mit Pavone Espresso, die viele geforderte Funktionen zur Webintegration und Workflowsteuerung mitbringt, die Voraussetzungen für XP erfüllt.

2.1 Groupwareprojekte und XP

Bei Groupware-Produkten handelt es sich um sozial eingebettete Systeme und hierbei speziell um Systeme, die eine Mensch-Maschine-Mensch-Interaktion ermöglichen sollen. Aus diesem Grunde und aus der Kritik an sequenziellen Vorgehensmodellen fordern TEUFEL ET AL. die Einhaltung von drei Prinzipien für die Entwicklung derartiger Systeme:[4]

- Benutzerinvolvierung,
- Iteratives Vorgehen und
- Multidisziplinarität.

Es soll nun gefragt werden, inwieweit die von BECK vorgeschlagenen Praktiken geeignet sind, diese Prinzipien zu erfüllen.

Im XP wird die Benutzerinvolvierung einerseits dadurch realisiert, dass ein Kunde Teammitglied sein soll, anderseits durch die Realisierung des sog. Planning Games. Ersteres bedeutet, dass ein „wirklicher“ Kunde mit dem Team zusammensitzen soll, er muss in der Lage sein, Unklarheiten bei den Anforderungen beseitigen können sowie detaillierte Prioritäten festlegen können.

Das Planning Game dient der Planung der Iterationen. Es sollen für die kommenden Iterationen Entwicklungsressourcen und Kundenwünsche in Einklang gebracht werden. Hierfür werden die Anforderungen der Anwender in User Storys zusammengetragen, die dann die Grundlage für die Projektplanung bieten. User Storys sind rudimentäre Anwendungsfälle und beschreiben einen Wunsch für ein zu realisierendes Feature und werden normalerweise auf Karteikarten geschrieben, dort geändert und zusammen mit den Benutzern immer wieder

durchgesprochen. Zum einen dient der Inhalt der Karten dazu, den Programmierern eine Schätzung des Realisierungsaufwands zu ermöglichen und noch nicht im ersten Schritt zu realisierende Anforderungen im Auge zu behalten, um diese möglicherweise in künftigen Releases zu verwirklichen; auf der anderen Seite sollen diese Karten die Diskussion zwischen Anwendern und Programmierern fördern. Programmierer schreiben selbst keine Storys, denn nur wenn die Anwender diese Storys eigenständig schreiben, sind es auch wirklich ‚ihre Storys'. Programmierer dürfen den Anwendern nur etwas helfen.

Durch Mitgliedschaft eines Kunden im Team und durch das Planungsspiel kann die zukünftige Akzeptanz des Groupware-Produktes bereits bei der Entwicklung gefördert werden. Die Kunden und zukünftigen Anwender partizipieren am Entwicklungsprozess; trotzdem sind die Rollen klar verteilt. Der Kunde entscheidet im Planungsspiel mittels der User Storys über Umfang, Prioritäten und Release-Zeiten. Die Entwickler schätzen den Aufwand, wägen Konsequenzen ab und erstellen eine detaillierte Zeitplanung.

Auch die Praktik des XP, bereits schnell ein erstes kleines Release fertigzustellen, das dann in iterativen Entwicklungszyklen weiterentwickelt wird, kann sich als geeignet erweisen, die Benutzerpartizipation zu fördern und die Akzeptanz rechtzeitig zu sichern. Dadurch, und auch wenn im weiteren Entwicklungsprozess sehr kurze Zyklen eingehalten werden, kann das Vertrauen des Kunden gestärkt werden, es stellen sich kurzfristig Erfolgserlebnisse ein, und durch ein unmittelbares Feedback werden Anpassungen an die Anforderungen leichter möglich. Wird so bewusst mit noch unvollständigen Anforderungen begonnen, können im Laufe des Entwicklungsprozesses Defizite erkannt und die erkannten Lücken schrittweise gefüllt werden.

Kurze Iterationen haben nicht nur den Vorteil, dass der Anwender sich bereits früh in der Phase der Entwicklung mit dem System auseinandersetzt, sondern auch, dass das System im anschließenden Produktivbetrieb als lebendig und veränderbar begriffen wird.

BECK empfiehlt, bereits früh ein funktionierendes Kernsystem in den Produktivbetrieb zu stellen. D.h. es werden zunächst einige User Storys umgesetzt; andere, zunächst noch nicht realisierte Storys können später implementiert werden. Hierbei spielen nicht nur Zeit- und Kostenüberlegungen eine Rolle. Es ist auch von Relevanz, dass solche Anforderungen umgesetzt werden, mit denen Quick Wins erzielt werden können. Die zu realisierenden Anforderungen sollten so gewählt werden, dass der spätere Benutzer nicht überfordert wird,

---

4 Vgl. Teufel u.a. 1995 /Gruppenarbeit/, S. 116ff.

aber in der Applikation sofort einen Wert für seine tägliche Arbeit erkennen kann. Insbesondere bei Groupware kann es wichtig sein, dass im System nicht gleich vollständige Prozesse abgebildet werden. Vielmehr sollte die Koordinationsleistung durch das System in ausgewählten Bereichen eingeführt und schrittweise erhöht werden, damit der Benutzer sich an die geänderten Arbeitsprozesse gewöhnen kann.

Auch in diesem Sinne ist dann XP als evolutionäres Modell zu verstehen: Evolutionär heißt hier, dass der Entwicklungsprozess „Neues hervorbringt, dabei auf schon Bekanntem aufbaut, Modifikationen vornimmt und wenn sich das entstandene Neue in seiner vorhandenen Umgebung bewähren muss, woraus ggf. Modifikationen und Anpassungen resultieren.“[5]

Hierdurch wird dann auch dem „Gesetz von der ständigen Änderung“ und dem „Gesetz vom kontinuierlichen Wachstum“ (LEHMAN) Rechnung getragen.[6] Er weist darauf hin, dass Programme kontinuierlich adaptiert werden müssen, da sie sonst an Angemessenheit verlieren. Zudem ist es nötig, den funktionalen Inhalt eines Programms kontinuierlich zu erhöhen, um die Zufriedenheit und Akzeptanz der Benutzer über die Lebenszeit des Programms sicherzustellen. Dies scheint sichergestellt, weil bei XP die inkrementelle Planung zu einem umfassenden Plan führt, der sich evolutionär entwickelt.

2.2 Lotus Notes und XP

Viele der Praktiken und Voraussetzungen von XP sind mit Anwendungen auf Basis von Lotus Notes gut zu vereinbaren. Dies sind insbesondere kleine Releases, einfacher Entwurf, Refactoring, gemeinsames Codeeigentum und kleines Entwicklerteam: [7] Durch Vorlagen und Replikationsmechanismen in Lotus Notes ist es möglich, viele kleine Releases ohne großen Overhead bei Installation und (weltweitem) Rollout zu den Anwendern zu bringen, um so schnell auf Anwenderwünsche reagieren zu können. Ein einfacher Entwurf wird durch die grafischen Entwicklungsoberflächen von Lotus Notes R5 und Pavone Espresso begünstigt, außerdem fällt das Verstehen von existierenden Applikationen und damit auch das Refactoring leichter. Durch den gemeinsamen Zugriff auf eine Vorlage oder eine Datenbank wird das gemeinsame Codeeigentum bei kleinen Entwicklergruppen unterstüzt. Repositories, die sich transparent in Lotus Notes integrieren, ermöglichen das Entwickeln in größeren Entwicklergruppen.

---

5 Hesse, Dahme 1997 /Software-Entwicklung/, S. 3.

6 Vgl. zu den Gesetzen Lehman 1980 /Laws/.

7 Für eine Übersicht über Praktiken und Voraussetzungen vgl. Reißing 2000 /Extremes Programmieren/.

Es gibt aber auch Voraussetzungen, die nicht offensichtlich gegeben sind. So muss zunächst noch geprüft werden, ob mit Lotus Notes automatisierte Tests durchführbar sind und ob anzunehmen ist, dass Änderungskosten nicht übermäßig mit der Nutzungsdauer steigen.

2.2.1 Änderungskosten bei später Berücksichtigung von Funktionen

Die erste Voraussetzung, die hier genauer diskutiert werden soll, ist der höchstens logarithmische Verlauf der Softwareänderungskosten. Traditionelle Softwareentwicklung geht davon aus, dass während der Anforderungsanalyse möglichst präzise und exakt allen Eventualitäten der Softwarenutzung Beachtung geschenkt werden muss, damit nicht später in diese Phase zurückgesprungen werden muss. Dahinter steckt die Annahme, dass die Softwareänderungskosten mit fortschreitendem Entwicklungsprozess exponentiell steigen. BECK hält dem entgegen, dass dies vielleicht einmal so war, aber durch Fortschritte bei Programmiersprachen, Praktiken und Tools nicht mehr gegeben ist. Er baut XP auf der Annahme auf, dass sich die Änderungskosten asymptotisch mit der Zeit entwickeln.[8] Dadurch können ohne Bedenken Erweiterungen und neue Features in das nächste und übernächste Release verschoben werden. Folgende Faktoren flachen den Kostenverlauf ab:

- Einfaches Design, d.h. es werden nur solche Features implementiert, die auch wirklich genutzt werden.
- Automatisierte Tests sollten sicherstellen, dass der Entwickler schnell und mit beruhigend großer Wahrscheinlichkeit feststellen kann, dass der Code, den er geändert hat, immer noch funktionsfähig ist.
- Erfahrung beim Ändern des Designs wird schon in der Entwicklungsphase gesammelt. Dadurch verringert sich die Angst vor einer späteren Erweiterung.[9]

Hierbei handelt es sich um eine entscheidende Praktik im XP-Vorgehensmodell. Deshalb muss entschieden werden, ob diese Bedingungen für die Softwareentwicklung in dem beschriebenen Projekt gegeben oder zumindest mit genügend hoher Wahrscheinlichkeit anzunehmen sind. BECK beruft sich bei der Begründung auf seine Erfahrungen. Hier soll versucht werden, anhand der Erfahrungen am Institut für Wirtschaftsinformatik in Marburg zu argumentieren. Die Praxis hat gezeigt, dass viele der Änderungs- und Erweiterungsanforderungen, die von den Anwendern existierender Anwendungen kamen, sich

---

8 Vgl. Beck 1999 /Extreme Programming Explained/, S. 21ff.
9 Vgl. Beck 1999 /Extreme Programming Explained/, S. 24f.

in kurzer Zeit mit Lotus Notes realisieren ließen. Ein großer Teil war nach maximal zwei Tagen implementiert, und eine Woche später war die neue Funktion getestet, dokumentiert und von den Benutzern akzeptiert.[10] Dies wurde durch das einfache Design der vorhandenen Applikationen gefördert und dadurch, dass die Entwickler mit dem Code der Applikationen durch Refactoring vertraut waren. Wenn ein Entwickler in Lotus Notes ein Formular um ein Feld ergänzt, einen View im Layout ändert oder die Felder um Funktionalität bereichert, selbst Datentypen von Feldern ändert, so ist dies möglich, ohne große Nebenwirkungen erwarten zu müssen. Unterstützt wird dies durch kurze Turnaround-Zeiten: der Entwickler kann das Feature sofort ausprobieren.

Automatisierte Tests wurden bisher nicht eingesetzt – vielleicht auch, weil die Applikationen selten viel Code enthielten. Ein großer Teil der Arbeit beschränkte sich auf das Anpassen von Layouts und das Hinzufügen von Formeln. Die Struktur des Codes wird durch die einzelnen Events eines Feldes oder eines Formulars vorbestimmt. Nur selten wird Lotus-Script-Code benötigt, der sich aber dann meist auf das aktuelle Formular oder den aktuellen View beschränkt. Die Erfahrung zeigt, dass auch ohne automatisierte Tests ein stabiler Betrieb und eine Weiterentwicklung ohne gravierende Fernwirkungen gewährleistet war. Damit kann angenommen werden, dass die Kosten für das Ändern und Hinzufügen von Funktionen weder exponentiell im Entwicklungsverlauf steigen noch besonders hoch sind und somit diese Voraussetzung für Extreme Programming gegeben ist.

2.2.2 Automatisierte Testfälle

Die zweite Voraussetzung für Extreme Programming sind automatisierte Testfälle. Ein Lotus-Notes-basiertes Anwendungssystem lässt sich in drei Arten von Programmteilen aufgliedern, die jeweils andere Anforderungen an automatisierte Testfälle stellen:

- Programmcode im klassischen Sinne: Textblöcke, die die Programmlogik in Form von Funktionen beinhalten.
- Design, Formularlogik, Makros und Workflows: Der Benutzer klickt sich durch ein Formular und im Hintergrund läuft verteilt auf verschiedenen Rechnern Logik ab. Benutzerrechte spielen dabei eine große Rolle.

---

[10] Der Administrator und Entwickler ist seit Oktober 1999 studentische Hilfskraft bei Prof. Hasenkamp am Institut für Wirtschaftsinformatik in Marburg.

- Basissystem (Hardware und Systemsoftware): Tests müssen komponentenübergreifend sein (Beispiel Mailrouting: Übergabe der Mails von einem externen Rechner, Routing durch Lotus Notes, Weiterverarbeitung durch Agenten, Workflow über Formulare und Formeln im Notes-Client).

Tests für den ersten Teil werden sich vergleichsweise einfach realisieren lassen, weil diese unabhängig von der übrigen Applikation getestet werden können. Die Testfälle lassen sich z. B. durch zusätzlichen Code nach dem eigentlichen Programmcode realisieren. Standardisiertes Vorgehen erscheint damit möglich. Beim zweiten Teil stellt sich dies schwieriger dar: automatische funktionale Tests des Notes-GUI erweisen sich als schwierig und sind ohne weitere Software nicht zu realisieren. Eine immer wieder gestellte Frage im Iris Cafe[11] ist, welche Software dafür geeignet sei: Antwort ist, dass sich das Notes-GUI wegen seiner besonderen Funktionen kaum automatisiert testen ließe. Gute Erfahrungsberichte waren nicht zu finden. Auch Anfragen bei anderen Lotus-Notes-Enwicklern und bei Lotus in Cambridge (MA) führten nicht zu konkreten Hinweisen bezüglich automatisierter Testtools. Deshalb wurde versucht, in dieser Kategorie zunächst nur mit Testfällen auf Papier auszukommen und entsprechende Codeteile klein zu halten.[12] Für den dritten Teil wird es sehr individuelle Tests geben, z. B. in Form von Unix Shellscripts.

Somit lässt sich feststellen, dass sich die Voraussetzungen nach BECK für XP in einer Lotus-Notes-Entwicklungsumgebung nicht vollständig einhalten lassen.[13] Deshalb war es notwendig, diesen Bereich genau zu beobachten, um ein Defizit frühzeitig erkennen zu können. Es ist anzunehmen, dass das Fehlen einer automatischen Testmöglichkeit die Größe von Lotus-Notes-Applikationen beschränkt, die mit XP entwickelt werden können.

2.3 Ist XP trotzdem möglich?

Die Diskussion in den letzten Abschnitten hat gezeigt, dass die Voraussetzungen für Extreme Programming für Lotus Notes mit Ausnahme des automatisierten Testens gegeben sind und dass diese Einschränkung bei dem Umfang des Projekts kein KO-Kriterium darstellt. Die Prüfung der Voraussetzungen hat außerdem gezeigt, dass einfaches Design und Refactoring

---

11 Treffpunkt der Notes-Entwickler unter http://www.notes.net.

12 Die Aussage „... seit Mitte der siebziger Jahre existiert eine geordnete Fülle von Methoden und Hilfsmitteln zur systematischen und rationellen Prüfung von Programmen“ (vgl. Belli 1998 /Systematische Prüfung/, S. 338) ist zumindest in Bezug auf die Hilfsmittel für Lotus Notes nicht gegeben.

13 Wobei dies jedoch kein Lotus-Notes-spezifisches Problem ist. Alle Systeme, die plattformübergreifend funktionieren und GUI-lastig sind, werden dieses Problem haben.

für leicht wartbaren Code wichtig sind. Dieser leicht wartbare Code bietet die Grundlage für kontinuierliche Verbesserung und lebendige Evolution des Systems. Deshalb wurde Extreme Programming als Systementwicklungsmodell für dieses Projekt gewählt. Unter Einbeziehung weiterer theoretischer Grundlagen zum Thema CSCW und zur Systementwicklung wurde ein Prototyp entwickelt. Mit den Anwendern wurden User Storys gesammelt und verschiedenen Stufen des Entwicklungsprozesses zugeordnet. Als einzige weitere Entwicklerdokumentation wurde ein einfaches Flowchart bei allen Sitzungen diskutiert und weiterentwickelt.

3 Lessons Learned aus dem durchgeführten Projekt

Nachdem die Entwicklung des Prototyps nun abgeschlossen ist, muss geprüft werden, inwieweit sich die Erwartungen in den verschiedenen Bereichen erfüllt haben.

3.1 Technische Sicht: Die Plattform Lotus Notes und Pavone Espresso

Die Erwartungen im Bereich Lotus Notes R5 waren, dass das System für die Zeitschrift eine zuverlässige Plattform zur Verfügung stellt. Diese Erwartungen wurden sowohl für die Entwicklungsphase als auch für die Nutzung erfüllt. Durch ein Server-Cluster konnte eine hohe Verfügbarkeit gewährleistet werden. Zur besseren Workflowunterstützung wurde auf Pavone Espresso Enterprise Office, eine Open Source Software für Lotus Notes, zurückgegriffen. Da Pavone Espresso viele Funktionen bereitstellt, die nur noch angepasst werden mussten, konnte den Anwendern bereits nach zwei Wochen eine Applikation mit hoher Funktionalität präsentiert werden. Dies legt die Vermutung nahe, dass Open Source Anwendungen einen guten Ausgangspunkt für XP-Projekte bilden, da nicht erst lange Zeit entwickelt werden muss, bevor eine brauchbare Kernapplikation verfügbar ist. So bekommen die Entwickler sehr früh eine Rückmeldung von den Anwendern, wie dies bei XP gefordert wird, und die Diskussion kann beginnen. Für die Entwickler ist es möglich, die Anwender zu begeistern und wegen der schon teilweise vorprogrammierten Strukturen in der Open Source Software zu Projektbeginn Quick Wins zu realisieren.

Die Änderungskosten von Lotus-Notes-Applikationen im Allgemeinen und von Pavone Espresso im Besonderen haben sich als verhältnismäßig gering herausgestellt. So konnten kleinere und größere Änderungen, die sich nach der Fertigstellung des ersten Prototypen ergaben, ohne großen Overhead vorgenommen werden. Auch wenn die Funktionen zum automatisierten Testen fehlen (und die Entwickler sie vermissen), lässt sich trotzdem ein stabiler Betrieb erreichen.

3.2 Organisatorische Sicht

Vergegenwärtigt man sich den Entwicklungsprozess nach XP, so sollen die Anforderungen der Anwender über den Kundenvertreter im Entwicklungsteam und über die User Storys eingebracht werden, die von den Anwendern im Planungsspiel diskutiert werden. Als Kundenvertreter im Team wurden Mitglieder der Redaktion der Zeitschrift

Wirtschaftsinformatik am Lehrstuhl von Prof. Hasenkamp gewählt. Die User Storys wurden gesammelt, indem der Entwickler die einzelnen Anwender (Herausgeber, Redaktion, Verlag, Setzerei), die über das Bundesgebiet verteilt sind, besuchte. Dieses Verfahren war aus mehreren Gründen nicht ganz XP-konform: Da die Anwender an verschiedenen Orten und in verschiedenen Organisationen arbeiten, gab es zuwenig Diskussion zwischen ihnen. Damit war der Anteil des Entwicklers an den User Storys zu hoch. Dadurch waren Lernprozesse und benutzergetriebene Veränderungen nur eingeschränkt möglich. An diesem Punkt zeigte sich, dass man hier nicht ohne Moderation auskommen kann. Die Diskussion zwischen den Anwendergruppen musste später nachgeholt werden; sie dauert noch an. Es ist aber abzusehen, dass die Schnittstellen zwischen den beteiligten Organisationseinheiten in befriedigendem Maße gestaltet werden können. Wäre diese Abstimmung früher erfolgt, so hätte der Softwareentwicklungsprozess deutlich beschleunigt werden können.

Sicherlich kann XP unter idealen Bedingungen viele Aufgaben des klassischen Projektmanagements wie Kosten- und Zeitüberwachung entbehrlich machen. Gleichzeitig tritt aber bei gruppenorientierten Anwendungssystemen die organisatorische Dimension in den Vordergrund. Es zeigte sich, dass das Projekt immer wieder Veränderungen im Arbeitsprozess und in der Arbeitsverteilung voraussetzte bzw. nach sich zog. Hier ist ein aktives Change Management auf der Ebene des Projektmanagements notwendig, um den Entwicklern den Rücken für die fachlichen und technischen Probleme freizuhalten.

Darüber hinaus ist einer der zentralen Aspekte von Extreme Programming, dass man eine Kernapplikation immer weiter ausbaut und dass jede dieser Stufen voll funktionsfähig bleibt. Daher erscheint es konsequent, die verschiedenen Stufen auch sofort in Produktion zu nehmen, um die Benutzer zur Auseinandersetzung mit der Software zu zwingen. Es stellte sich aber als schwierig heraus, die Anwender davon zu überzeugen, gleich die erste Version produktiv einzusetzen. Im weiteren blieb länger die Entscheidung offen, welches Release in Produktion genommen werden sollte. Auch dies ging möglicherweise einher mit der Tatsache, dass zu Beginn des Projekts kein aktives Change Management statt fand. Also zeigte sich auch hier, dass ein Projektmanagement nicht entbehrlich war.

4 Abschließende Projektbetrachtung

Dieses Projekt stellt einen großen Schritt in die richtige Richtung dar: Die Prozesse wurden analysiert und dokumentiert. Ein lauffähiger Prototyp eines Anwendungssystems für Herausgeber und Redaktion existiert. Diese Analyse der Workflows der Zeitschrift wird als Basis für die Weiterentwicklung der Organisation und Prozesse der Zeitschrift dienen.

Da die Erfahrungen mit Extreme Programming – bis auf die genannten Einschränkungen im organisatorischen Umfeld – positiv waren, wird es für die Begleitung der Software in Produktion und beim laufenden Einsatz weiter verwendet werden. Da in Zukunft mehrere Entwickler an dem Produkt arbeiten werden und nicht immer gewährleistet ist, dass sie alle an einem physisch gleichen Ort und zur gleichen Zeit arbeiten, wird die Zukunft zeigen, wie sich diese Abweichung von XP auf das Ergebnis auswirkt.

Literatur

Balzert 1996 /Lehrbuch der Software-Technik/

Balzert, Helmut, *Lehrbuch der Software-Technik. Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*, Berlin: Spektrum, 1998.

Beck 1999 /Extreme Programming Explained/

Beck, Kent, *Extreme Programming Explained: Embrace Change*, Reading (MA): Addison Wesley Longman, 1999.

Belli 1998 /Systematische Prüfung/

Belli, Fevzi, *Methoden und Hilfsmittel für die systematische Prüfung komplexer Software*, Informatik Spektrum 21 (1998), S. 337-346.

Cremers u. a. 1998 /PoliTeam/

Cremers, Armin; Kahler, Helge; Pfeifer, Andreas; Stiemerling, Oliver; Wulf, Volker, *PoliTeam – Kokonstruktive und evolutionäre Entwicklung einer Groupware*, Informatik Spektrum 21 (1998), S. 194-202.

Hesse, Dahme 1997 /Software-Entwicklung/

Hesse; Wolfgang; Dahme, Christian, *Editorial: Evolutionäre und kooperative Software-Entwicklung*, Informatik Spektrum 20 (1997), S. 3-4.

Lehmann 1980 /Laws/

Lehman, M. M., *Programs, Life Cycles, And Laws of Software Evolution,* IEEE, 68 (1980), S. 1069-1076.

Reißing 2000 /Extremes Programmieren/

Reißing, Ralf, *Extremes Programmieren*, Informatik-Spektrum 23 (2000) 2, S. 118-121.

Stahlknecht, Hasenkamp 1999 /Einführung Wirtschaftsinformatik/

Stahlknecht, Peter; Hasenkamp, Ulrich, *Einführung in die Wirtschaftsinformatik*, Berlin, Heidelberg: Springer, 9. Auflage, 1999.

Teufel u.a. 1995 /Gruppenarbeit/

Teufel, Stefanie u.a., *Computerunterstützung für die Gruppenarbeit*. Bonn: Addison Wesley, 1995.