Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

Erfahrungen mit XP

Doris Schmedding, Ingrid Beckmann
LS Software-Technologie, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund
{Doris.Schmedding, Ingrid.Beckmann}@udo.edu

Zusammenfassung

In einem Software-Praktikum hat ein Studierendenteam versuchsweise unter XP gearbeitet, während drei Gruppen einer phasenorientierten, modellbasierten Vorgehensweise folgten. Wir stellen Ergebnisse unsere Beobachtungen und Befragungen vor und ziehen ein Fazit für unsere weitere Arbeit.

1 Einleitung

Das eigene Ausprobieren aktueller Trends in der Softwaretechnik bietet Lehrenden die Chance, nicht nur ganz neue Erfahrungen mit den neuen Techniken zu gewinnen, sondern auch die bisher praktizierten Vorgehensweisen zu verbessern, indem Konzepte des neuen Vorgehens mit den bisher praktizierten und bewährten Verfahren verknüpft werden. Die gründliche Auseinandersetzung mit dem extrem anderen kann viele nützliche Erkenntnisse liefern.

E**x**treme Programming (XP) [Beck00] ist in den neunziger Jahren aus der Praxis als Gegensatz zur dokumentationslastigen Vorgehensweise bei der Softwareentwicklung entstanden, wie sie typischerweise heute an den Universitäten gelehrt und gelernt wird. Während die iterative und inkrementelle Vorgehensweise XP weitgehend auf Dokumentation verzichtet, soll in der konventionellen Vorgehensweise das Verständnis des Entwicklerteams von der Aufgabe und ihrer Lösung durch ein gemeinsam erstelltes, ausführlich dokumentiertes Modell erreicht werden, das heute meist in UML [BRJ99] notiert wird. Bei XP dient der Code selbst als Dokumentation.

Dem UML-basierten Vorgehen liegt die Vorstellung zugrunde, dass, wenn man das Problem nur gut genug analysiert und die Lösung genau genug spezifiziert, die Umsetzung des Modells in ein Programm ganz einfach wird und sich fast von allein ergibt. Es werden mächtige Werkzeuge eingesetzt, die aus Modellen Code-Rahmen erzeugen, in denen nur wenige Zeilen Code ergänzt werden müssen. Bei XP dagegen ist der Stellenwert der Programmierung als Entwicklungstätigkeit und die Qualität des erzeugten Codes besonders hoch. Gute Teamarbeit und die Integration des Kunden in den Entwicklungsprozess werden als Grundlage für erfolgreiche Projekte angesehen.

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

Um die Vor- und Nachteile alternativer Vorgehensweisen bei der Softwareentwicklung klarer erkennen zu können, wurde in studentischen Projekten im Rahmen des Software-Praktikums (SoPra) an der Universität Dortmund und auf der Informatica Feminale (IF), der Bremer Sommerschule für Informatikerinnen (*www.informatica-feminale.de*), mit XP experimentiert. Auf der IF wurde mit 10 Teilnehmerinnen ein einwöchiges XP-Projekt durchgeführt. Das Software-Praktikum ist eine Pflichtveranstaltung im Informatik-Grundstudium, in dem die Studierenden im Team von 6 bis 8 Studierenden entweder ein Semester oder sechs Wochen lang in einer Blockveranstaltung Softwareentwicklungs-Projekte durchführen. Insbesondere hat uns als Lehrende die Frage interessiert, wie auf sinnvolle Weise die XP-Konzepte einführt werden können, wie Programmieranfänger damit zurechtkommen und welche Techniken von XP besonders nutzbringend sind.

Im Anschluss werden zunächst ähnliche Projekte zur Einführung von XP vorgestellt, unser Versuchsrahmen beschrieben und unser Einstieg in XP erläutert. Dann wird genauer auf die für die Lehre wichtigen Merkmale von XP eingegangen, der Unterschied zur UML-basierten Vorgehensweise herausgestellt und unsere Ergebnisse in den Experimenten mit XP vorgestellt. Aus den positiven und negativen Erfahrungen werden Konsequenzen für die weitere Ausbildung im Software-Praktikum gezogen.

2 Die Einführung in XP

Newkirk und Martin beschreiben in [NeMa01] sehr anschaulich ein reales Pilotprojekt zur Einführung von XP in einer Softwarefirma. Keine der beteiligten Personen, alle erfahrene Softwareentwickler, hatte zuvor in einem echten XP-Projekt gearbeitet. Die lebhaften Schilderungen der positiven Erfahrungen und Fehler haben unsere Neugier geweckt, XP selbst auszuprobieren.

Auch an den Universitäten wird mit XP experimentiert. Williams und Upchurch [WiUp01] diskutieren, welchen positiven Einfluss die Konzepte von XP in der Softwaretechnik-Ausbildung haben könnten. Der tatsächliche Einfluss auf den Wissensstand der Studierenden lässt sich schlecht messen, aber ebenso wie wir konnten sie feststellen, dass die im XP-Kurs erzeugten Programme eine hohe Qualität besitzen.

Lippert et al. [LRWZ01] beschreiben ebenfalls den Einsatz von XP in einem Grundstudiums-Praktikum. Zwei Praktikumsgruppen, die beide nach der XP-Vorgehensweise arbeiteten, werden miteinander verglichen. Die Projektteams unterscheiden sich in erster Linie dadurch, wie viel Einfluss die Betreuer auf den Ablauf des Projekts genommen haben. Entsprechend mehr oder weniger chaotisch lief der Entwicklungsprozess ab.

In [MSK04] berichten die Autoren, welche Konflikte bei der Simulation betrieblicher Abläufe in einer Lehrveranstaltung auftreten können. Als ein Ergebnis der beschriebenen XP-Praktika an der Universität Bonn ist ein Rollenkonzept für die Lehre von XP an der Universität entstanden.

Angeregt durch die Beispiele wollten wir eigene Erfahrungen mit XP sammeln. Die überwiegend positiven Erfahrungen auf der Informatica Feminale haben uns ermutigt,

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

direkt im Anschluss XP auch im Software-Praktikum auszuprobieren. Das Praktikum fand als sechswöchige ganztägige Blockveranstaltung statt. Eine von vier Arbeitsgruppen, bestehend aus 6 Studierenden, führte ihr zweites Projekt nach der XP-Vorgehensweise durch, während die anderen nach einer UML-basierten Vorgehensweise [Schm01] arbeiteten. Das erste Projekt führten alle Gruppen nach der UML-basierten Vorgehensweise durch, die in der dem Praktikum vorangehenden Softwaretechnik-Vorlesung gelehrt wird. Dass nur eins der vier Praktikumsteams XP ausprobieren durfte, hängt mit dem großen Betreuungsaufwand in einem XP-Projekt zusammen. Ein XP-Coach, in diesem Fall die Gruppenbetreuerin, muss immer anwesend bzw. sofort erreichbar sein. Wissen, auch über den Entwicklungsprozess, wird in direkter Kommunikation vom Coach weitergegeben. Außerdem wollten wir die XP-Gruppe bei der Arbeit beobachten.

Zur Einführung der XP-Konzepte wurde in den beiden Lehrveranstaltungen jeweils zu Beginn ein XP-Spiel durchgeführt, um den iterativen Ablauf eines XP-Projekts kennen zu lernen und die Begriffe und die Rollen einzuüben, wie es z.B. in [AuMi01] beschrieben ist. Dieser Einstieg hat sich sehr bewährt und kann nur weiterempfohlen werden. Auf der Informatica Feminale hatten die Teilnehmerinnen sich außerdem durch Literaturstudium und durch Kurzvorträge in die Konzepte von XP eingearbeitet. Im Software-Praktikum haben wir nach dem Prinzip „learning by doing“ gearbeitet. Refactoring wurde vermittelt, indem nach der ersten Iteration mit Hilfe eines Beamers gemeinsam der Code verbessert wurde. Dabei wurde das Tool RefactorIt eingesetzt. Über Test-First hatten wir einen kurzen Vortrag vorbereitet, danach haben wir es gemeinsam am Beamer ausprobiert. Auch dabei haben wir mit Tool-Unterstützung gearbeitet. JUnit und RefactorIt lassen sich in die eingesetzte Entwicklungsumgebung TogetherJ integrieren.

3 Erfahrungen mit XP im Gegensatz zum UML-basierten Vorgehen

Wir stellen einige unserer Erfahrungen vor, die wir bei Beobachtungen und bei Befragungen der TeilnehmerInnen mit Hilfe von Fragebögen gewonnen haben. Dabei wurden wir von der Abteilung Organisationspsychologie der Universität Dortmund unterstützt. Besonders interessierte uns in unseren Experimenten der Einfluss bestimmter Konzepte, die teilweise im Gegensatz zu der bisher praktizierten UML-basierten Vorgehensweise [Schm01] stehen, auf den Lernprozess. In der Industrie werden auch viele andere Vorgehensmodelle, z.B. das V-Modell, erfolgreich eingesetzt, die mit dem hier beschriebenen UML-basierten Vorgehen die zentrale Stellung des Modells und die Wichtigkeit der erzeugten Dokumente gemeinsam haben.

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

3.1 Paararbeit

Bei der Paararbeit wird immer zu zweit an einem Rechner gearbeitet, eine tippt, ein anderer kontrolliert, fragt nach und liefert Ideen. Bereits während der Implementierung findet ein erstes Code-Review statt. So sollen Fehler vermieden und durch bessere Lesbarkeit und Verständlichkeit soll die Qualität des Codes gesteigert werden. In einer Lehrveranstaltung fördert Paararbeit mit ständig wechselnden Partnern das Lernen voneinander.

Als Konsequenz davon, dass durch so genannte Stories beschriebene Funktionalität realisiert wird, wobei viele Klassen des Systems verändert werden müssen und alle EntwicklerInnen abwechselnd an allen Programmteilen arbeiten, gehört der gesamte Code im XP-Projekt allen EntwicklerInnen, die ihn nicht nur lesen, sondern auch ändern dürfen. XP wird wegen dieses Konzepts und wegen der besonderen Bedeutung der direkten Kommunikation in erster Linie für kleinere Projekte mit etwa 10 EntwicklerInnen empfohlen [Beck00].

Die Paararbeit hat sowohl auf der Informatica Feminale als auch im Software-Praktikum wie auch bei Lippert [LRWZ01] gut geklappt, obwohl einige männliche Studierende anfangs Vorbehalte äußerten. Auf der Informatica Feminale konnten wir beobachten, dass ein Team, das in Paararbeit arbeitet, problemlos neue Mitglieder integrieren kann. Unser XP-Projektteam setzte sich täglich etwas anders zusammen. Das Ausscheiden eines Mitglieds konnte leicht verkraftet werden, da es keine ausgeprägten Spezialisten gab.

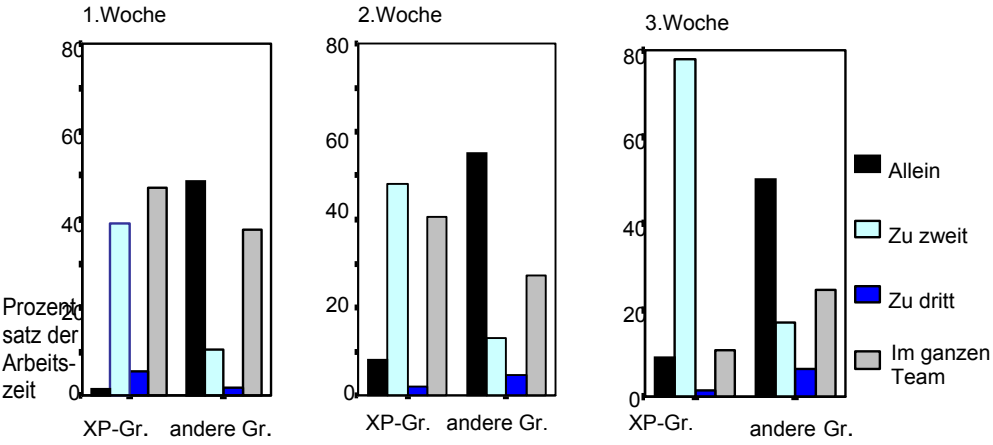

Abb. 1.: Paararbeit versus Einzelarbeit

Abbildung 1 zeigt als ein Ergebnis der Sopra-Befragung, dass die XP-Gruppe überwiegend in Paararbeit gearbeitet hat, während die anderen Gruppen überwiegend allein gearbeitet haben. Der große Anteil der Arbeit im gesamten Team insbesondere am Anfang des Projekts in der ersten und auch noch in der zweiten Woche ist bei der XP-Gruppe auf die Diskussion der Aufgabenstellung, die Planung der Iterationen und

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

die angeleitete Einarbeitung in die XP-Konzepte zurückzuführen. Diese Diskussionen über die Aufgabenstellung und die Projektsitzungen zur Organisation der Arbeit finden sich auch in den anderen Gruppen, die etwa ein Viertel ihrer Arbeitszeit im ganzen Team gearbeitet haben. Auch die UML-Gruppen haben mehr als 10 Prozent ihrer Zeit zu zweit gearbeitet.

3.2 Kommunikation und Teamklima

Die XP-Vorgehensweise ist darauf ausgerichtet, dass durch eine angenehme Arbeitsatmosphäre die gute Zusammenarbeit im Team und damit auch die Qualität des erstellten Produkts gefördert werden. Die Kommunikation unter den EntwicklerInnen wird als besonders wichtig für den Erfolg eines Projekts angesehen. In einer Lehrveranstaltung besteht der Reiz einer Vorgehensweise, die sich die Verbesserung des Teamklimas zum Ziel setzt, darin, durch ein gutes Teamklima auch ein gutes Lernklima zu erreichen.

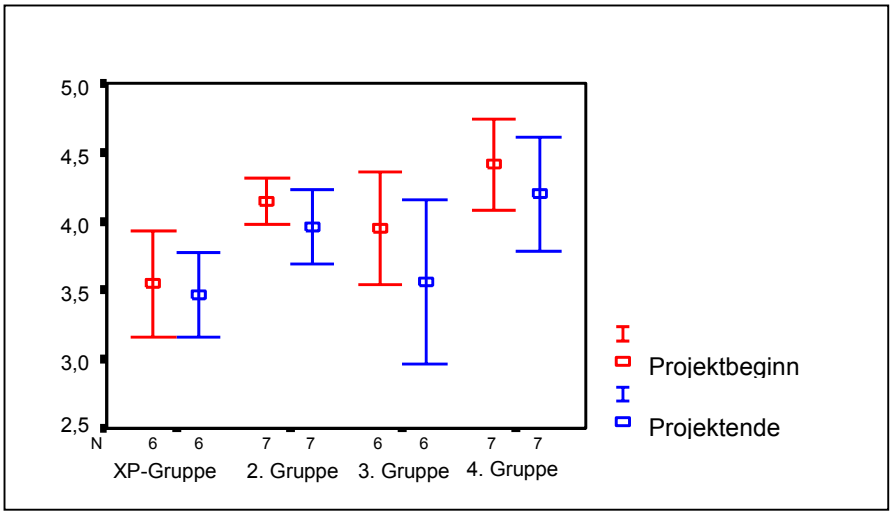

Abb. 2.: Teamklima in den Gruppen (links bei Projektbeginn, rechts bei Projektende)

In unserem Sopra-Experiment wurden zur Messung des Teamklimas Auszüge aus einem standardisierter Fragebogen verwendet, der erprobt ist und auch in Betrieben eingesetzt wird. Abbildung 2 zeigt, dass alle Gruppen ihr Teamklima zu Beginn des Projekts (Zeitpunkt 1) mit gut (>3,5) bewerten. Im Vergleich zu anderen Arbeitsteams in unserer Branche ist das ein guter Wert. Ausgerechnet in der XP-Gruppe (1. Gruppe) war das Arbeitsklima am schlechtesten, was wir vor der Befragung nicht ahnten. In dieser Gruppe waren die Vorkenntnisse besonders heterogen, was das kooperative Arbeiten belastete. In allen Gruppen zeigt sich am Ende des Projekts ein verschlechtertes Klima. Mit einer Verringerung des Mittelwerts um 0,08 fällt die Verschlechterung in der XP-Gruppe am geringsten aus. Die Standardabweichung

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

nimmt bei allen Gruppen außer der XP-Gruppe zu, hier verringert sich der Wert. Die Teammitglieder beantworteten die Fragen homogener als zu Beginn. Da unsere Datenbasis so klein ist, lässt sich aus dieser Beobachtung aber keine allgemeine Erkenntnis ableiten.

3.3 Kleine Iterationen und ständige Integration versus Modellierung und Dokumentation

Anstelle von aufwändiger Dokumentation in Form einer möglichst formalen Beschreibung wird durch ständig wechselnde Partner bei der Arbeit erreicht, dass das Wissen über die Inhalte und den Fortschritt des Projekts innerhalb des Entwicklerteams verbreitet wird. Die Betonung der Kommunikation gegenüber der Dokumentation steht eher im Gegensatz zu den bisherigen Lehrzielen im Grundstudium, wo viel Wert auf formal saubere Beschreibungen von Informatik-Inhalten gelegt wird.

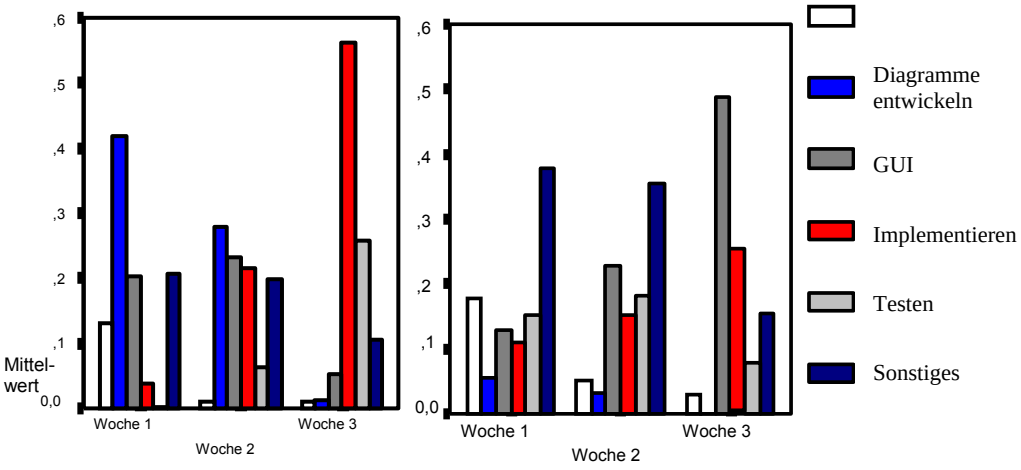

Abb. 3.: Tätigkeiten der Gruppen, links UML-Gruppen, rechts XP-Gruppe

Im iterativen XP-Prozess wird durch eine konsequente Beschränkung auf das Wesentliche ein komplexes Problem einfacher lösbar, und es entsteht sehr schnell eine erste funktionierende Version des Programms. Durch ständige Integration neuer Funktionalität wird das Programm in kleinen Iterationen schrittweise ausgebaut. Bei einer UML-basierten Vorgehensweise muss in den frühen Phasen viel Zeit für die Modellierung aufgewendet werden. Es besteht dabei die Gefahr, dass trotz hervorragender Modellierung nach Ablauf der Projektzeit kein lauffähiges Programm vorliegt. Bei der XP-Vorgehensweise entsteht bereits in der ersten Iteration eine vorzeigbare Version des Programms. Das führt frühzeitig zu Erfolgserlebnissen und trägt zur Motivation bei.

Abbildung 3 zeigt deutlich, wie unterschiedlich die verschiedenen Entwicklungstätigkeiten zu Buche schlagen. Während die UML-Gruppen (links), die einem wasserfallartigen Prozess [Schm01] folgen, zunächst Diagramme erstellen und

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

dann implementieren sowie testen, implementiert die XP-Gruppe von Anfang an. Die drei Wochen Projektzeit wurden in drei Iterationen aufgeteilt. Am Ende der ersten Woche existierte eine erste, allerdings winzig kleine Version des Programms, während die anderen nur Diagramme vorzeigen konnten. Die Entscheidung, welche Stories sich sinnvoll in einer ersten Iteration realisieren lassen, konnten die Studierenden in der XP-Gruppe allerdings nicht ohne Hilfe treffen. Sie hatte zu wenig Erfahrung, die Konsequenzen einer Story-Auswahl zu überblicken und die benötigte Zeit dafür einzuschätzen.

Die UML-Gruppen haben besonders zu Projektbeginn viel Zeit für die Modellierung verwendet. In den ersten beiden Wochen des Projekts wurden 42 bzw. 29 Prozent der Arbeitszeit für das Entwickeln der Diagramme aufgewendet. In der letzten Woche haben sich die Gruppen fast ausschließlich mit Implementieren und Testen beschäftigt. Die XP-Gruppe dagegen hat von Anfang an implementiert und getestet. Der Aufwand für die Tests war zumindest in den ersten beiden Wochen, als das XP-Vorgehen noch diszipliniert verfolgt wurde, höher als für die Implementierung.

Außerdem ist die Gefahr, den Überblick über das entstehende Programmsystem zu verlieren, bei XP ohne umfassendes Design relativ groß. UML-Diagramme können durchaus helfen, die Struktur des Systems und Abläufe zu verdeutlichen. Während wir auf der Informatica Feminale auf Wunsch der Teilnehmerinnen völlig auf UML-Diagramme verzichtet haben, haben wir im Software-Praktikum bewusst UML-Diagramme (Klassen- und Sequenzdiagramme) eingesetzt, wenn wir gemerkt haben, dass Abläufe und Zusammenhänge nicht allen klar waren. Wie weit man bei der Softwareentwicklung auf die Modellierung verzichten will und kann, hängt im Wesentlichen von der Erfahrung der EntwicklerInnen und von der Komplexität des Problems ab. AnfängerInnen hilft ein wenig Modellierung beim Verständnis der Aufgabe und ihrer Lösung.

3.4 Codequalität

Es leuchtet ein, dass der Qualität des Codes eine besondere Bedeutung bei XP zukommt. Der Code gehört allen und muss jederzeit für alle EntwicklerInnen verständlich sein. Er ist bereits während der Programmentwicklung einem ständigen Wartungsprozess unterworfen. Fester Bestandteil der XP-Vorgehensweise ist das „Refactoring“, die Verbesserung des existierenden Codes. Es reicht nicht aus, dass das Programm läuft und der erzeugte Code die Spezifikation erfüllt, vielmehr soll der erzeugte Code auch von hoher Qualität sein. Martin Fowler beschreibt in seinem Buch [Fowl00] verschiedene Kategorien von „schlecht riechendem“ Code und zeigt Wege, wie die Mängel beseitigt werden können. Die Programmierausbildung an den Universitäten konzentriert sich im Wesentlichen darauf, Sprachkonstrukte und ihre Anwendung vorzustellen und einzuüben. Der erzeugte Code wird hauptsächlich dahingehend beurteilt, dass das Programm die funktionalen Anforderungen erfüllt. Die Qualität des Codes bleibt weitgehend unberücksichtigt.

Unter XP wurde besserer Code erzeugt. Ein Vergleich des Programmcodes der XP-Gruppe mit dem Programmcode der anderen Gruppen ergab, dass die XP-Gruppe

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

kürzeren, leichter lesbaren und verständlicheren Code geschrieben hat als die anderen Gruppen [Bckm04]. Paararbeit und Refactoring scheinen die Lesbarkeit des Codes positiv zu beeinflussen. Wir konnten durch Messungen belegen, dass die XP-Gruppe z.B. sehr viel weniger duplizierten Code als die anderen Gruppen erzeugt hat [Bckm04]. Duplizierter Code ist besonders problematisch, weil er schlecht wartbar und bei Änderungen besonders fehleranfällig ist. Eine ganze Reihe der von Fowler [Fowl01] vorgeschlagenen Refactorings beschäftigen sich mit der Beseitigung von dupliziertem Code. Der massive Einsatz von Werkzeugen wie z.B. GUI-Buildern führt zu Code, der sehr „geschwätzig“ und schwerfällig ist. Er enthält sehr viel Redundanz, unnötige Kommentare und ist wenig elegant. Ein typisches Problem ist, dass die generierten Namen nicht durch sprechende Bezeichner ersetzt werden. Anstatt mit Vererbung zu arbeiten, wird bei ähnlichen Klassen der Code kopiert und abgewandelt. Das ist in der Wartungsphase äußerst problematisch, wenn alle Stellen gesucht werden müssen, an denen Änderungen vorgenommen werden müssen.

3.5 Testen

Die Bedeutung des Testens in der Softwareentwicklung wird bei XP besonders betont. Bevor eine Komponente implementiert wird, wird nach dem Test-First-Prinzip der Test dafür geschrieben. Die Spezifikation einer Komponente erfolgt also durch die Definition des Tests, den sie erfüllen muss. Es wird nur genau so viel implementiert, dass der Test durchläuft, und keine Zeile mehr. Kommen neue Anforderungen hinzu, wird zunächst das Testprogramm erweitert, das dann natürlich nicht mehr durchläuft. Danach die Klasse ergänzt, bis der Test wieder durchläuft. So soll vermieden werden, dass mehr implementiert wird, als unbedingt notwendig ist.

Im traditionellen Prozess ist Testen nach Abschluss der Implementierung bei Studierenden besonders unbeliebt, JUnit dagegen motiviert durch schnelle Erfolgserlebnisse. Aus Abbildung 3 lässt sich ablesen, dass die XP-Gruppe tatsächlich viel getestet hat, anfangs mehr, als sie implementiert hat, während sich die UML-Gruppen erst ab der zweiten Woche mit der Implementierung und dann hauptsächlich in der dritten Woche mit dem Testen beschäftigt haben.

Nach einer gewissen Umgewöhnungszeit waren auch die PraktikumsteilnehmerInnen in der Lage, nach dem Test-First-Prinzip zu arbeiten. Wir konnten beobachten, dass Testen sogar Spaß machen kann, wenn der „grüne Balken“ erscheint. Von der XP-Gruppe wurde im Software-Praktikum mehr Zeit für das Testen als für das Implementieren aufgewendet. Für Test-First von GUI-Klassen ist uns allerdings keine Lösung eingefallen.

3.6 Kundenorientierung

Starke Kundenorientierung durch direkte Integration des Kunden als aktives Projektmitglied stellt ein weiteres Merkmal von XP dar. Durch die iterative Vorgehensweise kann leicht auf sich ständig ändernde Anforderungen des Kunden reagiert werden. Die Bedeutung des Kunden im Entwicklungsprozess lässt sich in

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

studentischen XP-Projekten nur mit hohem Aufwand simulieren [MSK04]. Als Kunde ist ein Lehrender nach XP auch aktiv in den Entwicklungsprozess eingebunden, er formuliert Abnahmetests und bestimmt bei der Iterationsplanung mit. Ein Kunde, der vom Veranstalter simuliert wird, drängt auf die Fertigstellung des möglichst umfangreichen Produkts. Dieser Rollenkonflikt zwischen Lehrendem, der auf die korrekte Einhaltung der Abläufe und Konzepte achten muss, und dem simulierten aktiven Kunden wird auch von Mügge, Speicher und Kniesel [MSK04] bestätigt.

Bei der phasenorientierten Vorgehensweise geht man davon aus, dass zu einem bestimmten Zeitpunkt alle Anforderungen des Kunden erhoben und dokumentiert sind. Dann kann mit der eigentlichen Entwicklung begonnen werden. Das lässt sich von Lehrenden in studentischen Projekten zwar leicht realisieren, indem von ihnen entsprechende Dokumente vorbereitet werden, ist aber eine unrealistische Sicht der Wirklichkeit, denn reale Kunden ändern ständig während der Entwicklung ihre Anforderungen. Agile Vorgehensweisen wie XP sind hier den wenig flexiblen, auf Dokumenten basierenden Vorgehensweisen überlegen.

Aber Projekte mit realen Kunden lassen mit vielen studentischen TeilnehmerInnen nur unter unvertretbar hohem Aufwand durchführen.

3.7 Spezialistentum versus Wissenstransfer

Bisher konnten wir in den Studenten-Projekten ein ausgeprägtes Spezialistentum beobachten. Aus der Sicht der Gruppe ist es äußerst effizient, Spezialisten z.B. für das grafische User Interface auszubilden und den erfahrensten Programmierer möglichst ungestört implementieren zu lassen, während andere das Benutzungshandbuch schreiben. Aus der Sicht von Lehrenden ist hierbei problematisch, dass sowohl die Arbeit als auch der Wissenserwerb ungleich verteilt sind. Das Ziel einer Lehrveranstaltung im Grundstudium muss aber sein, dass alle TeilnehmerInnen alle Inhalte zumindest ansatzweise lernen. Bei XP ist die Gefahr des extremen Spezialistentums durch die Paararbeit bei ständig wechselnden Partnern und Aufgaben nicht groß.

XP hat sich durchaus als sehr effizient erwiesen. Das XP-Team hat weniger Stunden als die übrigen Gruppen gearbeitet, aber bei kürzerem Code ein im Funktionsumfang vergleichbares Produkt entwickelt, obwohl konsequent zu zweit an einem Rechner gearbeitet wurde. In Paararbeit wird sehr konzentriert gearbeitet. Beide Partner müssen ständig bei der Sache sein. XP verlangt eine Vereinbarung von festen Arbeitszeiten, damit Paararbeit im Team funktionieren kann.

Zurzeit führen wir eine intensive Studie zur Paararbeit durch. Dabei beobachten wir die gesamte Arbeit der Teams, die sich unterschiedlich organisiert haben. Die Kommunikation untereinander und die Weitergabe von Wissen sind in den Paararbeitsteams sehr viel umfangreicher als bei Nicht-Paararbeitsteams.

Klaus-Peter Löhr · Horst Lichter (Hrsg.) (2005): Software Engineering im Unterricht der Hochschulen, SEUH 9, Aachen, dpunkt.verlag, Heidelberg

4 Fazit

XP ist für programmierunerfahrene EntwicklerInnen schwierig und verlangt viel Disziplin. Es fällt AnfängerInnen schwer zu entscheiden, welche Stories sich sinnvoll zu einem funktionalen Kern für eine Iteration zusammenfassen lassen. Das Schätzen der für die Realisierung einer Story benötigen Zeit ist für unerfahrene EntwicklerInnen schwierig. Ein leichtgewichtiger Prozess bietet außerdem die Gefahr, ins Chaos abzugleiten, wenn nicht diszipliniert gearbeitet wird. Ähnliche Beobachtungen schildern auch Lippert et al. [LRWZ01]. Wir konnten beispielsweise beobachten, dass bei Zeitdruck auf Test-First verzichtet wurde, was fatale Folgen hatte. VeranstalterInnen sollten dann im Interesse der Lernenden Einfluss nehmen und auf die Einhaltung des Prozesses dringen. Ein geplant vorliegender, phasenorientierter Prozess [Schm01] bietet Anfängern, die ja Softwareentwicklung gerade erst lernen sollen, Halt und Übersicht.

Ein XP-Projekt ist betreuungsintensiv. VeranstalterInnen von XP-Projekten müssen mehrere Rollen einnehmen, die nur schwer vereinbar sind. Als Lehrende müssen sie insbesondere auf die saubere Umsetzung der Konzepte achten, auch wenn das Projektziel gefährdet ist. Daneben müssen die Lehrenden im XP-Projekt auch die Rolle der aktiven KundInnen übernehmen, die immer anwesend oder zumindest schnell erreichbar sein sollen, falls es keine echten Kunden gibt.

Literatur

[AuMi01] K. Auer, R. Miller: Extreme Programming Applied – Playing to Win. Addison Wesley, 2001.

[Beck00] K. Beck: Extreme Programming explained: Embrace Chance. Addison Wesley, 2000.

[Bckm04] I. Beckmann: Erkennen von Code-Mängeln zum Refaktorisieren. Diplomarbeit am Fachbereich Informatik der Universität Dortmund, 2004.

[BRJ99] G. Booch, J. Rumbaugh, I. Jacobson: The Unified Modeling Language – User Guide. Addison Wesley, 1999.

[Fowl00] M. Fowler: Refactoring, oder: Wie Sie das Design vorhandener Software verbessern. Addison-Wesley, 2000.

[KSS04] C. Kopka, D. Schmedding, J. Schröder: Der Unified Process im Grundstudium – Didaktische Konzeption, Einsatz von Lernmodulen und Erfahrungen. In: DELFI 2004 – Die 2. Deutsche e-Learning Fachtagung Informatik, Paderborn, LNI, 2004.

[LRWZ01] M. Lippert, S. Roock, H. Wolf, H. Züllighofen: XP lehren und lernen. in: H. Lichter, M. Glinz (Editors): Software Engineering im Unterricht der Hochschulen, SEUH 7, Zürich 2001. dpunkt.verlag.

[MSK04] H. Mügge, D. Speicher, G. Kniesel: Extreme Programming in der Informatik-Lehre – Ein Erfahrungsbericht. In: P. Dadam, M. Reichert (Hrsg.): Informatik 2004 – Informatik verbindet, Proceedings der 34. GI-Jahrestagung, Band 2, Ulm, LNI, 2004.

[NeMa01] J. Newkirk, R. C. Martin: Extreme Programming in Practice. Addison Wesley, 2001.

[Schm01] D. Schmedding: Ein Prozessmodell für das Software-Praktikum. In: H. Lichter, M. Glinz (Hsg.): SEUH 2001. Software Engineering im Unterricht der Hochschulen. S. 87-97. Zürich. 2001. dpunkt.verlag.

[WiUp01] L. Williams, R. Upchurch: Extreme Programming for Software Engineering Education? In: 31st ASEE/IEEE Frontiers in Education Conference, Reno, NV, 2001.