

Experimente mit XP in der Lehre

Ingrid Beckmann und Doris Schmedding

Informatik 10 (Software-Technologie)
Universität Dortmund
ib@betu.de, Doris.Schmedding@udo.edu

Wir berichten über unsere Erfahrungen beim versuchsweisen Einsatz von XP in Software-Entwicklungspraktika an der Universität. Mit der Projektplanung nach XP waren unsere wenig programmiererfahrenen TeilnehmerInnen teilweise überfordert, aber insbesondere die Konzepte von XP zur Verbesserung des Codes und der Kommunikation im Team haben sich sehr bewährt. Auf diese wollen wir auch in modellbasierten Projekten nicht mehr verzichten.

1. Einleitung

Extreme Programming (XP) [Beck00] ist in den neunziger Jahren aus der Praxis als Gegensatz zur dokumentationslastigen, modellbasierten Vorgehensweise bei der Software-Entwicklung entstanden, wie sie typischerweise heute an den Universitäten gelehrt wird. Da bei XP der Code als Dokumentation dient, 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.

Um die Vor- und Nachteile alternativer Vorgehensweisen bei der Software-Entwicklung klarer erkennen, wurde in studentischen Projekten im Rahmen des Software-Praktikums (SoPra) an der Universität Dortmund und auf der Informatica Feminale, der Bremer Sommerschule für Informatikerinnen, mit XP experimentiert. Das Software-Praktikum ist eine Pflichtveranstaltung im Informatik-Grundstudium, in dem die Studierenden in Teams von 6 - 8 entweder ein Semester oder sechs Wochen lang in einer Blockveranstaltung Software-Entwicklungsprojekte durchführen. Insbesondere hat uns die Frage interessiert, wie die XP-Konzepte eingeführt werden und wie Programmieranfänger damit zurechtkommen. XP wird eigentlich für fortgeschrittene Programmierer empfohlen.

Newkirk und Martin beschreiben in [NeMa01] sehr anschaulich ein reales Pilotprojekt zur Einführung von XP in einer Software-Firma. Keine der beteiligten Personen, alle erfahrene Software-Entwickler, hatte zuvor in einem XP-Projekt gearbeitet. Die lebhaften Schilderungen der positiven Erfahrungen und Fehler haben unsere Neugier geweckt, XP selbst auszuprobieren. Williams und Upchurch [WiUp01] diskutieren, welchen positiven Einfluss die XP-Konzepte in der Software-Technik-Ausbildung an der Universität haben könnten. Der tatsächliche Einfluss auf den Wissensstand der Studierenden

lässt sich schlecht messen, aber wie wir stellten sie fest, dass die im XP-Kurs erzeugten Programme eine hohe Qualität besaßen. Lippert et al. [LRWZ01] berichten von zwei Praktikumsgruppen, die nach der XP-Vorgehensweise arbeiteten. Die Teams unterschieden sich dadurch, wie viel Einfluss die Betreuer auf den Ablauf des Projekts genommen haben. Entsprechend mehr oder weniger chaotisch lief der Entwicklungsprozess ab.

Angeregt durch die Beispiele wollten wir eigene Erfahrungen mit XP sammeln. Besonders interessierten uns bestimmte Konzepte, die teilweise im Gegensatz zu der bisher praktizierten UML-basierten Vorgehensweise [Schm01] stehen. Aus den positiven und negativen Erfahrungen in unseren Experimenten werden Konsequenzen für die weitere Ausbildung im SoPra gezogen und es wird aufgezeigt, wie sich bewährte XP-Konzepte in bestehende Prozesse integrieren lassen.

2. Merkmale von XP im Gegensatz zum UML-basierten Vorgehen

Bei der **Paararbeit** wird immer zu zweit am 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. Bisher konnten wir in den Projekten ein ausgeprägtes Spezialistentum, z.B. für das grafische User Interface, beobachten, das für die Gruppe äußerst effizient ist. In einer Lehrveranstaltung sollten aber alle TeilnehmerInnen alle Inhalte zumindest ansatzweise lernen. Auch in professionellen Teams ist die Wissensverbreitung im ganzen Team wichtig. Paararbeit fördert das Lernen voneinander. Da im XP-Projekt 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 allen, die ihn nicht nur lesen, sondern auch ändern dürfen.

Die XP-Vorgehensweise ist darauf ausgerichtet, dass 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 Projektes angesehen. 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.

Durch eine konsequente Beschränkung auf das Wesentliche wird ein komplexes Problem einfacher lösbar und eine erste funktionierende Version des Programms entsteht sehr schnell. Das führt frühzeitig zu Erfolgserlebnissen und trägt zur Motivation bei. Durch **ständige Integration** neuer Funktionalität wird das Programm in **kleinen Iterationen** schrittweise ausgebaut. Bei einer UML-basierten Vorgehensweise wird in den frühen Phasen viel Zeit für die Modellierung aufgewendet. Es besteht dabei die Gefahr, dass trotz hervorragender Modellierung am Ende kein lauffähiges Programm vorliegt.

Fester Bestandteil der XP-Vorgehensweise ist das „**Refactoring**“, die Verbesserung des existierenden Codes. Martin Fowler beschreibt in seinem Buch [Fowl00] verschiedene Kategorien von „schlecht riechendem“ Code und zeigt Wege, die Mängel zu beseitigen. Die Programmierausbildung an den Universitäten konzentriert sich im Wesentlichen darauf, Sprachkonstrukte vorzustellen und einzuüben. Der erzeugte Code wird hauptsächlich dahingehend beurteilt, dass das Programm die funktionalen Anforderungen erfüllt, während die Qualität des Codes weitgehend unberücksichtigt bleibt.

Die Bedeutung des **Testens** wird bei XP besonders betont. Vor der Implementierung einer Komponente wird der Test geschrieben. Die Spezifikation einer Komponente erfolgt also durch die Definition des Tests. Es wird nur genau soviel implementiert, dass der Test gerade erfüllt ist. Im traditionellen Prozess ist Testen bei Studierenden besonders unbeliebt, JUnit dagegen motiviert durch schnelle Erfolgserlebnisse.

3. Erfahrungen in den XP-Projekten

Die IF 2003 (www.informatica-feminale.de) bot die Chance, im Rahmen eines einwöchigen Kurses mit 10 erfahrenen Entwicklerinnen ein kleines XP-Projekt durchzuführen. Durch die überwiegend positiven Erfahrungen ermutigt, probierten wir XP auch im SoPra an der Universität Dortmund im Rahmen einer sechswöchigen ganztägigen Blockveranstaltung aus. Eine von 4 Arbeitsgruppen führte ihr zweites Projekt nach der XP-Vorgehensweise durch, während die anderen nach einer UML-basierten Vorgehensweise [Schm01] arbeiteten. Wir stellen einige Beobachtungen und Ergebnisse von Befragungen der TeilnehmerInnen vor. Dabei wurden wir von der Abteilung Organisationspsychologie der Universität Dortmund unterstützt.

Zur Einführung von XP wurden verschiedene Ansätze gewählt. Um den iterativen Ablauf eines XP-Projekts kennen zu lernen und die Begriffe und die Rollen einzuüben, wurde jeweils ein XP-Spiel [AuMi01] durchgeführt. Das hat sich sehr bewährt und kann nur weiterempfohlen werden. Auf der IF hatten die Teilnehmerinnen sich außerdem durch Literaturstudium und Kurzvorträge in die Konzepte von XP eingearbeitet. Im SoPra haben wir nach dem Prinzip „learning by doing“ gearbeitet. Refactoring wurde vermittelt, indem nach der ersten Iteration 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 mit Toolunterstützung ausprobiert. JUnit und RefactorIt lassen sich in unsere Entwicklungsumgebung TogetherJ integrieren.

XP hat sich als sehr effizient erwiesen. Das XP-Team hat weniger Stunden als die übrigen Gruppen gearbeitet, aber bei kürzerem Code ein vergleichbares Produkt entwickelt, obwohl konsequent in Paarbeit gearbeitet wurde. Da beide Partner ständig bei der Sache sein müssen, wird sehr konzentriert gearbeitet. Die UML-Gruppen haben viel Zeit für die Modellierung verwendet, in den ersten beiden Wochen 42 % bzw. 29 % der Arbeitszeit. 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 XP noch diszipliniert verfolgt wurde, höher als für die Implementierung.

Zur Messung des Teamklimas wurden Auszüge aus einem standardisierter Fragebogen verwendet. Alle Gruppen bewerteten ihr Teamklima zu Beginn des Projekts mit gut, was ein guter Wert ist. Ausgerechnet in der XP-Gruppe war das Arbeitsklima am schlechtesten, da die besonders heterogenen Vorkenntnisse die Kooperation belasteten. In allen Gruppen zeigt sich am Ende ein Verschlechterung, die in der XP-Gruppe am geringsten ausfällt. Die Standardabweichung nimmt bei allen Gruppen außer in der XP-Gruppe zu. Die Teammitglieder beantworteten die Fragen homogener als zu Beginn. Aufgrund unserer kleinen Datenbasis lässt sich daraus aber keine allgemeine Erkenntnis ableiten.

Die Paararbeit hat sowohl auf der IF auch im SoPra wie auch bei Lippert [LRWZ01] gut geklappt, obwohl einige männliche Studierende anfangs Vorbehalte äußerten. Auf der IF konnten wir beobachten, dass ein Team in Paararbeit problemlos neue Mitglieder integrieren kann. Das Ausscheiden oder Hinzukommen eines Mitglieds konnte leicht verkraftet werden, da es keine ausgeprägten Spezialisten gab. Mit XP wird besserer Code erzeugt. Ein Vergleich des Programmcodes der XP-Gruppe mit dem der anderen Gruppen ergab, dass die XP-Gruppe kürzeren und verständlicheren Code geschrieben hat [Bckm04]. Wir konnten durch Messungen belegen, dass die XP-Gruppe z.B. sehr viel weniger duplizierten Code als die anderen Gruppen erzeugt hat [Bckm04], der schlecht wartbar und bei Änderungen besonders fehleranfällig ist. Viele der von Fowler [Fowl01] vorgeschlagenen Refactorings beschäftigen sich mit der Beseitigung dieses Mangels. Der Einsatz von Werkzeugen wie z.B. GUI-Buildern führt zu sehr „geschwätzig“ und schwerfällig Code. Er enthält viel Redundanz und unnötige Kommentare. Ein typisches Problem ist, dass die generierten Namen nicht durch sprechende Bezeichner ersetzt werden. Anstatt mit Vererbung zu arbeiten, wird bei ähnlichen Klassen Code kopiert und abgewandelt.

Nach einer gewissen Umgewöhnungszeit waren die PraktikumssteilnehmerInnen 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 SoPra mehr Zeit für Testen als für Implementieren aufgewendet.

XP ist für unerfahrene EntwicklerInnen schwierig, insbesondere zu entscheiden, welche Stories sich sinnvoll zu einem funktionalen Kern für eine Iteration zusammenfassen lassen. Ein leichtgewichtiger Prozess bietet die Gefahr, ins Chaos abzugleiten. Ähnliche Beobachtungen schildern auch Lippert et al. [LRWZ01]. Unter Zeitdruck wird gerne auf Test-First verzichtet, was fatale Folgen hat. Deshalb ist ein XP-Coach unerlässlich. Die Gefahr, den Überblick über das entstehende Programmsystem zu verlieren, ist ohne umfassendes Design relativ groß. UML-Diagramme können durchaus helfen, die Struktur des Systems und Abläufe zu verdeutlichen. Während wir auf der IF völlig auf UML-Diagramme verzichtet haben, wurden im SoPra UML-Diagramme (Klassen- und Sequenzdiagramme) eingesetzt, wenn Abläufe und Zusammenhänge nicht allen klar waren.

4. Fazit

XP ist keine grundsätzlich neue Vorgehensweise, sondern eine Zusammenfassung von guten Praktiken [Beck00]. Deshalb scheint es uns legitim, in Grundstudiumsprojekten

auf schwierige Konzepte zu verzichten und Konzepte, die sich bewährt haben, in unseren Entwicklungsprozess zu übernehmen. Das sind insbesondere die Techniken zur Verbesserung des Codes und der Teamarbeit, wie Paararbeit, Refaktorisieren und JUnit-Tests.

Paararbeit lässt sich in jedem Entwicklungsprozess bei fast allen Tätigkeiten einsetzen. Wir haben gute Erfahrungen mit Paararbeit bei der Modellierung im UML-basierten Prozess und bei der Einarbeitung in neue Tools gemacht. So lassen sich Fehler vermeiden und man erhält besser lesbaren Code. Paararbeit scheint zwar zunächst aufwendig, ist aber durchaus effizient. Paararbeit wollen wir in weiter untersuchen, insbesondere interessiert uns das Rollenverhalten zwischen Driver und Observer, die Kommunikation zwischen den Partnern und die Wissensweitergabe im Team.

Die Maßnahmen zur Steigerung der Qualität des Codes stellen eine Bereicherung jeden Entwicklungsprozesses dar. Insbesondere Refactoring lässt sich in jeden Prozess integrieren. Wir haben es auch außerhalb von XP-Projekten mit Studierenden durchgeführt. Dabei sind Tools wie RefactorIt notwendig. Ebenso lässt Test-First sich leicht in einen modellbasierten Entwicklungsprozess integrieren lässt: Vor der Implementierung der Methoden erfolgt die Implementierung der Tests, die nach und nach ausgebaut werden.

Die aufgeführten Techniken verursachen zwar zunächst Mehraufwand, führen aber zu besserem Code. Ein gutes Modell und die Verwendung von Werkzeugen führen nicht zwangsläufig zu gutem Code. Wir wollen auch im UML-basierten Entwicklungsprozess auf viele Konzepte von XP nicht mehr verzichten. Sie lassen sich einfach und sinnvoll integrieren. Wie weit man bei der Software-Entwicklung 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.

Literatur

- [AuMi01] Ken Auer, Roy 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, in Arbeit.
- [Fowl00] M. Fowler: Refactoring, oder: Wie Sie das Design vorhandener Software verbessern. Addison-Wesley, 2000.
- [LRWZ01] M. Lippert, S. Roock, Henning Wolf, Heinz Züllighofen: XP lehren und lernen. in: Horst Lichter, Martin Glinz (Editors): Software Engineering im Unterricht der Hochschulen, SEUH 7, Zürich 2001
- [NeMa01] James Newkirk, Robert C. Martin: Extreme Programming in Practice. Addison Wesley, 2001.
- [Schm01] D. Schmedding: Ein Prozessmodell für das Software-Praktikum. In: Lichter, H. und Glinz, M. (Hsg.), SEUH /. 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.