Extremes Programmieren

Erschienen im Informatik-Spektrum 23(2), 2000, S. 118-121 als „Aktuelles Schlagwort“

Ralf Reißing
Universität Stuttgart
Institut für Informatik
Abteilung Software Engineering
Breitwiesenstr. 20-22
reissing@informatik.uni-stuttgart.de

1 Einleitung

Extremes Programmieren (Extreme Programming, XP) ist ein Prozessmodell für die objektorientierte Software-Entwicklung. Wie der Name andeutet, handelt es sich um ein code-zentriertes Prozessmodell, bei dem bestimmte Techniken in extremem Maße angewendet werden. Einige Annahmen und Techniken von XP stehen in starkem Widerspruch zum Software Engineering, das vom Capability Maturity Model und ISO 9000 geprägt ist. Diese Widersprüche führen zu lebhaften Diskussionen über diverse Aspekte von XP.

In diesem Artikel wird gezeigt, welche Ideen hinter XP stehen, aus welchen Techniken XP besteht und welche Voraussetzungen für eine Implementierung von XP gegeben sein müssen. Darüber hinaus werden auch Schwachpunkte und Grenzen des XP-Ansatzes beleuchtet.

2 Ansatz

Das extreme Programmieren wurde von Kent Beck, Ward Cunningham und Ron Jeffries entworfen und durch Erprobung in Projekten weiterentwickelt. XP ist gedacht für kleinere Projekte mit unklaren und sich immer wieder ändernden Anforderungen. Der Kunde hat gleichzeitig hohe Ansprüche an die Qualität der Software.

2.1 Praktiken

XP ist ein Prozessmodell, das sich durch eine Reihe von Techniken auszeichnet, die in der XP-Terminologie Praktiken heißen. Diese Praktiken sind nur in ihrer Gesamtheit sinnvoll. Das Weglassen wichtiger Praktiken kann zum Scheitern des XP-Ansatzes führen, da sich die Praktiken gegenseitig stützen. Keine der eingesetzten Praktiken ist wirklich neu; neu ist die Kombination und der „extreme“ Grad, in dem diese Praktiken eingesetzt werden.

Kleine Releases. Aufgrund der sich ständig ändernden Anforderungen wird ein hochgradig iterativer Entwicklungsprozess mit einen Release-Zyklus von ein bis drei Monaten angestrebt. Ein Release besteht aus mehrere Iterationen von ein bis drei Wochen Dauer. Die Iterationen wiederum zerfallen in Arbeitpakete mit einer Dauer von ein bis drei Tagen. Nach jeder Iteration kann der Kunde Abweichungen von seinem Wünschen feststellen und korrigieren. Das erste Release sollte bereits ein funktionierendes Kernsystem sein, das dann in weiteren Releases inkrementell ausgebaut wird.

Planungsspiel. Die Planung der Iterationen erfolgt im Planungsspiel. Ziel des „Spieles“ ist es, für die kommende Iteration auf Grund vorgegebener Spielregeln die verfügbaren Entwicklungsressourcen und die Kundenwünsche in Einklang zu bringen. Die Anforderungen werden

in Form sogenannter Storys niedergelegt. Das sind rudimentäre Anwendungsfälle, die auf eine Karteikarte geschrieben werden. Der Kunde versieht seine Storys mit Prioritäten. Er bestimmt, welche Storys in der kommenden Iteration zu implementieren sind und wann die Iteration abgeschlossen sein soll. Daraufhin geben die Entwickler Schätzungen ab, wie viel Aufwand zur Implementierung notwendig sein wird. Wahrscheinlich wird der Aufwand die verfügbaren Ressourcen übersteigen, so dass der Kunde einige Storys auf spätere Iterationen verschieben muss. Dies wird so lange wiederholt, bis ein Gleichgewicht erreicht ist. Auf diese Weise gelangt man zu einer relativ realistischen Planung für die kommende Iteration. Falls die Schätzungen der Entwickler daneben lagen, kann bei Bedarf nachverhandelt werden.

Tests. Automatisierte Tests spielen *die* zentrale Rolle bei XP; ohne sie ist XP nicht realisierbar. Da außer den Storys keine Spezifikation des Systems und seiner Bestandteile vorliegt, wird das Wissen über die gewünschte Funktion in Testfällen niedergelegt. Die Entwickler schreiben Tests für ihre Klassen (unit tests). Der Kunde entwickelt Testfälle für seine Storys (functional tests), die von einem Tester in Tests umgesetzt werden. Die Testfälle werden so implementiert, dass sich alle Tests jederzeit automatisch ausführen lassen.

Wichtig ist, dass die Entwickler *erst* die Tests schreiben und dann implementieren. Das zwingt sie dazu, eine rudimentäre Spezifikation der zu testenden Funktionen in Form von Fallbeispielen anzugeben. Sobald dann die Implementierung vorliegt, kann sie gegen die Tests geprüft werden.

Systemmetapher. Die Systemmetapher steht für die grundsätzliche Idee hinter der Architektur des Systems. Sie soll sowohl für die Entwickler als auch für den Kunden verständlich sein. Die Systemmetapher soll dazu dienen, einen Architekturentwurf zu ersetzen; stattdessen sollen sich die Entwerfer beim Entwerfen von der Systemmetapher leiten lassen.

Einfacher Entwurf. Beim Entwurf soll immer nach einer einfachen Lösung gesucht werden, die nur die momentan anstehenden Anforderungen abdeckt. Beck nennt das “the simplest thing that could possibly work”[1]. Zukünftige Erweiterungen sollen zunächst nicht berücksichtigt werden, da sich die Anforderungen ändern können, wodurch sich die zusätzlich eingebaute Flexibilität als nutzlos erweist. Sollten später Änderungen notwendig sein, wird der Entwurf refaktorisiert.

Refaktorisierung. Refaktorisierung dient zur Vereinfachung des Entwurfs eines bestehenden Systems unter Beibehaltung der Semantik. Dabei soll die Verständlichkeit und die Änderbarkeit des Codes verbessert werden. Da die gewünschte Semantik in den Tests niedergelegt ist, kann nach einer Refaktorisierung durch Test geprüft werden, ob dabei Fehler unterlaufen sind. (Näheres zum Vorgehen bei Refaktorisierung findet sich in [4]).

Laut Beck lässt sich beobachten, dass ein großer Teil der Dokumentation, die erstellt wird, mangels laufender Anpassung sehr schnell veraltet und daher gar nicht benutzt wird. Deshalb beschränkt XP die Dokumentation auf den Code selbst. Dieser muss daher in hohem Maße selbsterklärend sein. Die Selbsterklärungsfähigkeit soll vor allem aus der Struktur und der Namensgebung kommen. Wenn der Code an einer Stelle eines erklärenden Kommentars bedarf, sollte so refaktorisiert werden, dass der Code auch ohne Kommentar verständlich ist.

Programmieren in Paaren. Programmieren in Paaren (pair-programming) ist die zweite zentrale Praktik von XP. Entwurf, Codierung und Test werden von zwei Entwicklern gemeinsam durchgeführt. Während einer der beiden Entwickler programmiert, prüft der Partner den Code auf Schreibfehler und logische Fehler. Außerdem behält er andere Ziele im Auge, z. B. ob der Code zum Entwurf passt, der Entwurf verbessert werden kann oder ob zu dem Code noch Tests

fehlen. Auf diese Weise werden der Code, der Entwurf und die Tests einer kontinuierlichen Begutachtung unterzogen. Bei Bedarf tauschen die beiden Entwickler die Rollen. Die Paarbindung ist nicht fest, sondern man sucht sich für jedes Arbeitspaket einen geeigneten Partner aus. Die dynamische Paarbildung hat den positiven Nebeneffekt, dass sich das Wissen über alle Aspekte des Systems auf alle Entwickler verbreitet. Wenn ein Entwickler ausfällt, kann er daher relativ problemlos ersetzt werden.

Williams et al. [6] haben die Auswirkungen des Programmieren in Paaren empirisch untersucht. Ihrer Untersuchung zufolge nimmt der Entwicklungsaufwand im Vergleich zu allein arbeitenden Entwicklern nur leicht zu (um etwa 10%), während der Code besser verständlich ist und viel weniger Fehler aufweist.

Gemeinsames Code-Eigentum. Der entstehende Code gehört nicht einem bestimmten Entwickler, sondern allen Entwicklern im Projekt. Das bedeutet, dass jedes Entwicklerpaar jederzeit überall Änderungen vornehmen darf, um z. B. die Verständlichkeit des Codes zu verbessern. Dabei besteht natürlich die Gefahr, dass die Änderungen fehlerhaft sind. Daher muss das Paar nach jeder Änderung alle Tests laufen lassen.

Kontinuierliche Code-Integration. Neu entwickelter oder geänderter Code wird alle paar Stunden in die aktuelle Code-Basis integriert. Zur Integration dient ein dedizierter Integrationsrechner. Dort spielt ein Paar seine Änderungen ein und lässt danach alle Tests laufen. Treten dabei Fehler auf, muss das Paar seine Änderungen zurücknehmen oder dafür sorgen, dass alle Fehler behoben werden. Auf diese Weise ist immer ein lauffähiges System verfügbar. Außerdem zerfallen durch das Vorgehen die Arbeitspakete in kleine, überschaubare Teile.

40-Stunden-Woche. Das Programmieren in Paaren stellt hohe Ansprüche an die Partner. Beide müssen jederzeit hoch konzentriert bei der Sache sein. Das können sie aber nicht, wenn sie erschöpft sind. Daher werden bei XP geregelte Arbeitszeiten gefordert (die 40 Stunden sind dabei nur ein Richtwert). Überstunden, hier definiert als unfreiwillige Mehrarbeit, sollen nur in Ausnahmefällen gemacht werden.

Kundenvertreter im Team. Da keine echte Spezifikation vorhanden ist, gibt es viele Rückfragen an den Kunden. Daher muss ein Vertreter des Kunden für die Entwickler permanent verfügbar sein (on-site customer). Es soll sich um einen zukünftigen Anwender des Systems handeln. Der Kundenvertreter entwickelt die Testfälle für die funktionalen Tests.

Programmierrichtlinien. Um das Arbeiten in Paaren und das gemeinsame Code-Eigentum zu erleichtern, halten sich alle Entwickler an feste Programmierrichtlinien. Diese werden von den Entwicklern untereinander vereinbart und strikt eingehalten. Auf diese Weise erhält man einheitlichen Code, der von allen verstanden und geändert werden kann.

2.2 Voraussetzungen

Wie man den einzelnen Praktiken entnehmen kann, hat XP einige Voraussetzungen für eine erfolgreiche Implementierung:

1. Die Änderungskosten steigen höchstens logarithmisch mit der Zeit. Ansonsten wird der Ansatz des einfachsten Entwurfs durch die ständige Refaktorisierung zur Integration neuer Anforderungen zu teuer.

2. Das Management, alle Teammitglieder und der Kunde müssen sich auf die XP-Praktiken einlassen. Außerdem muss der Kunde in der Lage sein, einen qualifizierten Mitarbeiter abzustellen.
3. Das Entwicklerteam darf nicht zu groß sein. Bei 10-15 Entwicklern liegt die Obergrenze.
4. Die Entwickler sind an einem Ort konzentriert und haben die gleichen Arbeitszeiten, da sonst Kommunikation und Paarbildung erschwert sind.
5. Die Testfälle müssen automatisch und in kurzer Zeit ausführbar sein. Lange Übersetzungs- und Ausführungszeiten sind von großem Nachteil.

Beck empfiehlt eine schrittweise Einführung von XP. Es sollte immer nur eine Praktik auf einmal eingeführt werden, und zwar immer diejenige, die das momentan dringlichste Problem angeht [1]. Natürlich sind dabei die Abhängigkeiten der Praktiken untereinander zu berücksichtigen.

3 Bewertung

Wie bereits erwähnt mangelt es nicht an Kritik am extremen Programmieren (siehe z. B. [5]). Einige Punkte sollen hier herausgegriffen werden.

- Das Fehlen einer expliziten Spezifikation und einer Entwurfsdokumentation ist besonders kritisch. Es gibt zwar die Dokumentation in Form der Testfälle und des Codes, doch ist es zweifelhaft, ob diese allein auch für Entwickler ausreicht, die nicht Mitglied im XP-Team waren. Ohne die beteiligten Entwickler geht es vermutlich nicht.
- Das gemeinsame Code-Eigentum kann zu einigen Problemen führen. Die Entwickler müssen mangels eines Übersichtsdokuments den Entwurf als mentales Modell im Kopf haben. Ändert ein anderer Entwickler den Entwurf im Code, muss das mentale Modell angepasst werden, da sonst mit falschen Voraussetzungen codiert wird und dadurch Fehler entstehen. Außerdem kann es zu konkurrierenden Änderungen an denselben Klassen kommen, was zu Integrationsproblemen führt.
- Darüber hinaus ziehen Änderungen am Entwurf in der Regel auch Änderungen an den Tests nach sich. Da viele XP-Praktiken voraussetzen, dass die Tests korrekt sind, muss hier besonders sorgfältig gearbeitet werden. Ob die Begutachtung beim Programmieren in Paaren wirklich ausreicht, die Schwächen des Testansatzes zur Qualitätssicherung zu kompensieren, ist fraglich.
- Problematisch ist, dass XP bisher nur unzureichend dokumentiert ist, insbesondere wenn es um die konkrete Umsetzung geht. Zum Beispiel ist das Konzept der Systemmetapher reichlich nebulös. Es gibt auch keine Daten, die nachweisen, dass XP anderen Vorgehensweisen überlegen ist. Empirische Untersuchungen wurden bisher nicht durchgeführt.

Andererseits gibt es viele (Jubel-)Berichte aus Projekten, in denen XP eingesetzt wurde. Das bekannteste Projekt ist das C3-Projekt bei DaimlerChrysler [3]. Bei Beck [2] finden sich auch Berichte aus anderen Projekten. Aus den Berichten geht hervor, dass alle beteiligten Gruppen mit XP zufrieden sind. Entwickler berichten von hoher Motivation und Freude bei der Arbeit, das Management freut sich über die gute Termineinhaltung, und die Kunden begrüßen die frühe Verfügbarkeit eines funktionierenden Systems sowie die hohe Qualität.

Weitere Informationen zu XP finden sich im Referenzwerk von Kent Beck [1] und unter www.xprogramming.com. An der Diskussion teilnehmen kann man in den News-Gruppen

comp.object und comp.software-eng sowie in den Mailing-Listen xp-forum und extremeprogramming bei www.egroups.com.

Literatur

1. Beck, K.: Extreme Programming Explained: Embrace Change. Reading/MA. Addison-Wesley 1999
2. Beck, K.: Embracing Change with Extreme Programming. IEEE Computer 32(10), 1999, 70-77
3. C3 Team: Chrysler Goes to “Extremes”. Distributed Computing, Oktober 1998, 24-28
4. Fowler, M.: Refactoring: Improving the Design of Existing Code. Reading/MA. Addison-Wesley 1999
5. Wegener, H.: Extreme Ansichten: Für und Wider des Extreme Programming. iX 12/1999, 126-130
6. Williams, L.; Kessler, R.; Cunningham, W.; Jeffries, R.: Strengthening the Case for Pair-Programming. (Erscheint in IEEE Software; siehe auch http://www.cs.utah.edu/~lwilliam/Papers/ieeeSoftware.PDF)