Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Datenmanagement und -exploration
Prof. Dr. T. Seidl

Proseminar „Methoden und Werkzeuge"

eXtreme Programming

Benedikt Franz

Juni 2012

Betreuung: Prof. Dr. T. Seidl
Dr. P. Kranen

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Was ist eXtreme Programming

> *XP is a style of software development focusing on excellent application of programming techniques, clear communication, and teamwork which allows us to accomplish things we previosly could not even imagine.* [1]

eXtreme Programming, auch XP genannt, zählt zu den agilen Entwicklungsmethoden in der Softwareentwicklung. Agil bedeutet hier eine schnelle Reaktionsfähigkeit auf sich ändernde Anforderungen während des Entwicklungsprozesses. Es ist eine gute Prinzipien- und Techniken-Sammlung, um zeitnah und erfolgreich Software zu entwickeln, XP erfreut sich deshalb großer Beliebtheit.

In den neunziger-Jahren entstanden viele agile Ansätze (z.B. das Pair Programming), die Kent Beck zum Teil mitentwickelte und schlussendlich im Jahr 2000 in einem Buch [2] veröffentlichte. In den darauffolgenden Jahren gewann XP immer mehr Anerkennung und wurde auch aufgrund seiner provokant empfundenen Thesen sehr bekannt. Beck machte in der darauffolgenden Zeit sehr viele Erfahrungen und erweiterte seine vorgestellten Modelle stetig, bis er schließlich 2005 eine zweite, stark überarbeitete Fassung seines XP-Manifestes [1] auf den Markt brachte.

1.2 Beispiel - Entwicklung einer Mitgliederverwaltung: *Koala*

Die bestehende Mitglieder-Organisation mithilfe von Excel-Tabellen der Pfadfinder Westfalen sollte von einer webbasierten Mitgliederverwaltung abgelöst werden. Unser Team bestand aus drei Freunden, von denen einer Pfadfinder war. Er diente als direkter Ansprechpartner, falls Fragen zu den internen Abläufen, Gestaltungen oder ähnlichem auftauchten. Unbewusst wurden während der Entwicklung sehr viele XP-Verfahren verwendet, die im Nachhinein den Erfolg der Umsetzung in dem Projekt mit dem Namen *Koala* ausmachten.

Einige der zentralen Techniken werden im folgenden Kapitel genauer erläutert und nehmen jeweils einen kurzen Bezug zum Projekt *Koala*. Das dritte Kapitel fasst die Ausarbeitung zusammen und gibt einen kurzen Ausblick.

Kapitel 2

eXtreme Programming

Im folgenden Teil werden zunächst die Begriffe XP-Werte, -Prinzipien und verschiedene -Techniken beschrieben. Anschließend wird ein kurzer Vergleich zu traditionellen und verwandten Entwicklungsmethoden (*DXP*, *IXP* und *Scrum*) gezogen.

2.1 XP-Werte

eXtreme Programming beruht ursprünglich auf den vier Werten: *Einfachheit*, *Kommunikation*, *Feedback* und *Mut*. Nach der Überarbeitung (zweites Buch) ist der Wert *Respekt* hinzugekommen, siehe auch Abbildung 2.1. Diese Werte stellen die Grundlage für alle darauf aufbauenden Aktionen dar.

Alle Prinzipien und Techniken wurden im Zuge des überarbeiteten XP-Manifestes [1] neu formuliert und ergänzt, näher präzisiert und/oder vereinfacht. Auch zog Beck im Zuge seiner Erfahrungen der letzten Jahre neue Schlüsse und wollte frische Herausforderungen für bereits agierende XP-Teams setzen.

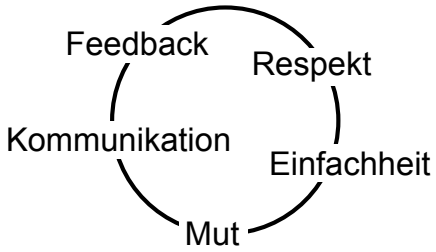

Abbildung 2.1: Die 5 XP-Werte

2.2 XP-Prinzipien

Aus den fünf zentralen XP-Werten haben sich 14 XP-Prinzipien entwickelt. Viele davon sind selbst-sprechend und werden in den Büchern [1] und [4] genauer beschrieben. Die Bedeutungsvollsten sind dabei: Gegenseitiger Vorteil, Menschlichkeit, Wirtschaftlichkeit, Verbesserung, Reflexion und Mannigfaltigkeit.

2.3 XP-Techniken

Wurden zunächst die ursprünglichen XP-Techniken nicht diversifiziert, unterscheidet Beck nun in seiner Neuauflage zwischen Primär- und Folgetechniken. In der Abbildung 2.2 wurden diese unter Anlehnung an Grafik [4, Abb. 3-3] zusammengestellt.

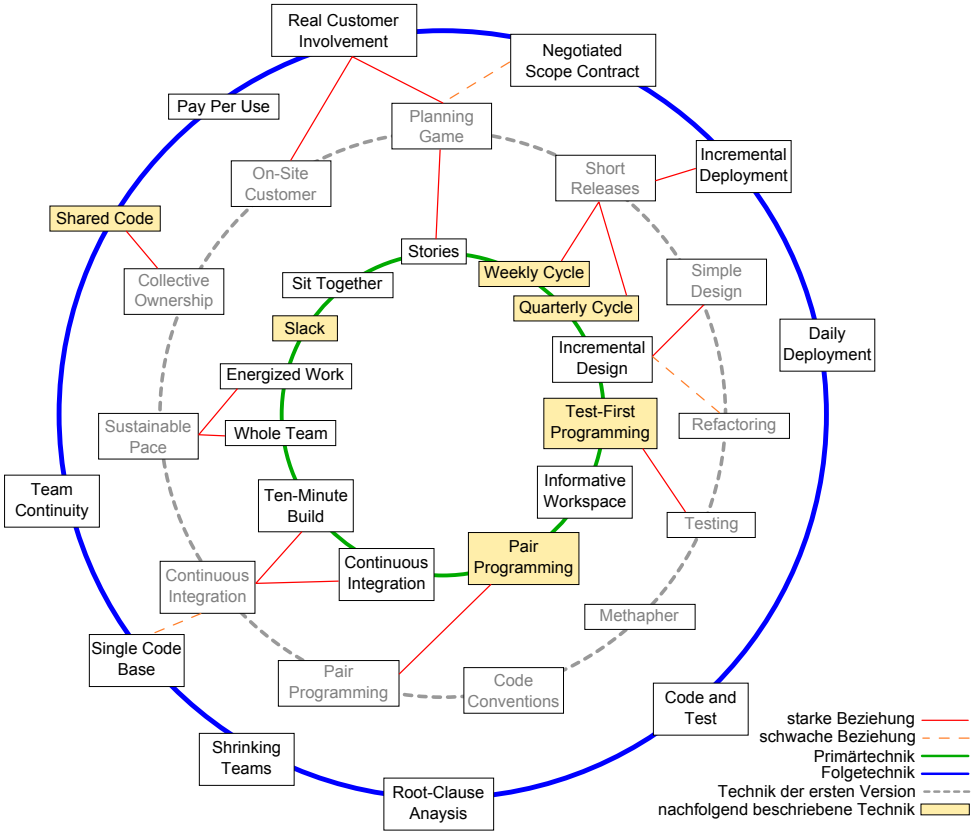

Abbildung 2.2: Die Techniken von XP

Gut zu erkennen ist, dass viele Techniken als Grundlage der neu vorgestellten Methodiken dienten. Die *Metapher* wurde entfernt, weil viele sie nicht verstanden und auch nicht angewendet haben.

Im Folgenden wird ein Teil der Techniken kurz beschrieben. Weitere Erläuterungen zu allen XP-Prinzipien und XP-Techniken sind unter anderem in den Büchern [1] und [4] zu finden.

Primärtechnik: Pair Programming Zwei Personen sitzen vor einem Bildschirm und arbeiten zusammen am selben Quellcode. Was sich im ersten Moment sehr unproduktiv anhört, ist in der Praxis durchaus empfehlenswert. Beide Charaktere sind gleichberechtigt und wechseln alle paar Minuten die Kontrolle, hier in Form der Tastatur. Durch die sofortige und unmittelbare Feedbackmöglichkeit können ebenfalls schnell Lösungs-, Verbesserungs- oder Änderungsansätze entdeckt werden.

Beim Projekt *Koala* wurde diese Technik sehr häufig in ähnlicher Weise eingesetzt. Zwar arbeiteten nicht alle an einem Bildschirm mit der gleichen Tastatur, aber man schaute sich häufig über die Schulter und entwickelte schnell Lösungen, sodass man kontinuierlich weiterarbeiten konnte. Nicht selten steigerte sich dadurch auch die Performance, sowohl auf bei der Geschwindigkeit der Entwicklung, als auch auf der technischen Seite. So konnte zum Beispiel durch gezielte Datenbankabfrage die Ergebnisse einfacher verarbeitet werden.

Ein großes Problem beim *Pair Programming* stellt die physische Präsenz beider Entwickler am gleichen Standort dar. Wie dieser Konflikt gelöst werden kann, wird später kurz im Abschnitt *DXP* beschrieben.

Primärtechnik: Test-First Programming Bereits von Anfang an beschreibt Beck das sofortige Testen der entwickelten Methoden als elementar wichtig. Spätestens mit der Neuauflage der XP-Techniken gilt nun die verpflichtende Grundlage, zunächst Test-Programme zu schreiben und erst danach die dazugehörigen Methoden zu implementieren. Dabei wird im Wesentlichen in Komponenten- und Akzeptanztests unterschieden.

- **Komponententests** dienen dazu, die entwickelten Klassen zu testen.
- **Akzeptanztests** sollen überprüfen, ob der Kunde das momentane System benutzen könnte und auch so akzeptieren würde.

Das Projekt *Koala* wurde von erster Minute an getestet. Sind Methoden ergänzt oder verändert worden, mussten alle damit verbundenen Dienste auf korrekte Funktionsweise getestet und ein aufgetretener Fehler sofort gesucht und behoben werden. Aufgrund der webbasierten Entwicklung war Becks Empfehlung des automatisierten Testens bei *Koala* allerdings nicht gegeben. Hier wurde die Wichtigkeit des regelmäßigen Testen allerdings sehr deutlich: Wenn unglücklicherweise ein Test vergessen wurde, konnte dies im Nachhinein zu einer langen Suche führen. Dies war durchaus zeitintensiver, als es ein kurzer Test gewesen wäre.

Primärtechnik: Quarterly Cycle Bei der *normalen* Softwareentwicklung ist es üblich, dass Projekt auszuliefern, sobald es alle nötigen Spezifikationen erfüllt. Bei XP sieht dies anders aus. Hier legt man Wert darauf, dem Kunden immer wieder ein bereits lauffähiges Teilstück abzuliefern. Ein Release dauert dabei ein Quartal (drei Monate), dann besteht die Möglichkeit das bis dahin entwickelte Projekt im Einsatzbereich zu testen. Dadurch kann sehr viel schneller entdeckt werden, ob eventuelle Missverständnisse aufgetreten sind.

Bei *Koala* fand diese Technik in entfernt ähnlicher Weise statt. Stets wurde nach Funktionserweiterung von unserem Pfadfinder-Team-Mitglied geprüft, ob diese den gedachten Anforderungen entsprechen. So konnte schnell auf eine eventuell missverständlich realisierte Methodik reagiert werden.

Primärtechnik: Weekly Cycle Ein Release unterteilt sich noch einmal in wöchentliche Iterationen auf. Am Ende einer Iteration werden dem Kunden die zuletzt implementierten Ergebnisse präsentiert und die Anforderungsumsetzung für die nächste Iteration geplant.

Primärtechnik: Slack Beim Slack handelt es sich um den Freiraum des Entwicklers, sich stets auf dem technologisch neusten Stand halten zu können. So ist es zum Beispiel möglich, dass neue Vorgehensweisen den Entwicklungsprozess vereinfachen. Hat ein Teammitglied allerdings keine Zeit sich regelmäßig zu informieren, könnte dieser für nachfolgende Projekte nicht ausreichend qualifiziert sein.

Bei *Koala* wurde jedem dieser Freiraum gelassen. Ab und zu konnte die Entdeckung von interessanten, neuen Umsetzungsmöglichkeiten die Realisierung um sehr viele Arbeitsstunden verkürzen. So konnten zum Beispiel

durch die Einführung von jQuery die Benutzeroberfläche intuitiver gestaltet und Eingabefehler direkt abgefangen werden.

Folgetechnik: Shared Code Die Technik des *Shared Code*s ist eine sehr wichtige Praxis bei XP. Keinem gehört der Quellcode und jeder darf ihn zu jeder Zeit bearbeiten. Dieses stellt aber auch gewisse Herausforderungen dar: Zum einen müssen alle Team-Mitglieder den Überblick über das Projekt haben, zum anderen werden ähnliche Qualifikationen benötigt. Eine resultierende Konsequenz ist, dass alle Entwickler gemeinsam für den Code verantwortlich sind. Tritt ein Fehler auf, wird dieser vom Finder behoben. Dabei wird empfohlen, den Verursacher freundlich darauf hinzuweisen.

Ein Projekt sollte bei XP möglichst unabhängig von den Entwicklern sein. Die Abhängigkeit von einem Entwickler wird als *Truck-Faktor* bezeichnet. Dieser gibt die Wahrscheinlichkeit des Projektscheiterns an, wenn der zugehörige Entwickler von einem Truck überfahren wird. Somit sollte dieser Wert generell möglichst gering ausfallen.

Zu manchen Zeitpunkten schien der Truck-Faktor bei uns *Koala*- Entwicklern ziemlich hoch. Dies lag daran, dass ab und an manche Funktionalitäten alleine entwickelt werden mussten. Da nicht an Kommentaren gespart wurde, stand es jedem offen, den Code zu verändern oder Fehler sofort zu beheben.

2.4 Vor- und Nachteile gegenüber traditionellen Methoden

Natürlich ist XP nicht das Allheilmittel, mit dem sich jedes Projekt in kurzer Zeit realisieren lässt. Aber es trägt ein gutes Stück dazu bei, agiler und fortschrittlicher zu handeln.

Vorteile Der wohl größte Vorteil ist die bereits genannte Agilität. XP ist durch seine kurzen Iterationen in der Lage, auf sich ändernde Anforderungen ebenfalls schnell reagieren zu können. Ist ein Änderungswunsch bei XP eher eine Kleinigkeit, so könnte dies zum Beispiel beim Wasserfallmodell eine komplette Projektneuplanung zur Folge haben.

Nachteile XP-Anwender müssen sich mit den Werten, Prinzipien und Techniken identifizieren können. Sollten sich Team-Mitglieder nicht verstehen oder die Vorgehensweisen nicht akzeptieren, wird XP eher ein Hindernis sein. Klassische Modelle betrifft dies oftmals nicht: Hier gibt es zumeist feste Zuständigkeiten und teilweise ist die einzige Kommunikation zwischen den Entwicklern die Schnittstellendefinition. Eine für XP notwendige gute Teamarbeit ist hier nicht zwingend erforderlich.

2.5 Verwandte Methoden

XP ist auch aus der Entwicklungssicht nicht immer vollständig oder fehlerfrei. Trotz der umfangreichen Prinzipien und Techniken gibt es zum Teil Diskrepanzen oder zu viele/wenige Einschränkungen. Dem entgegenzuwirken haben auch andere Entwickler XP für ihre Zwecke angepasst.

Im Folgenden gibt es einen knappen Überblick über zwei Erweiterungen von XP und einen kurzen Vergleich zu einer ähnlichen agilen Strategie namens *Scrum*.

DXP Bei vielen XP-Techniken müssen sich die Entwickler geographisch nah beieinander aufhalten, um zum Beispiel das Pair Programming erfolgreich auszuführen. Da die starke physische Nähe nicht immer gewährleistet werden kann, wurde Distributed eXtreme Programming entwickelt. Es beruht darauf, dass viele Techniken (zum Beispiel das Testen, Refactoring, Weekly Cycle) nicht zwangsläufig am selben Ort stattfinden müssen, sondern mit heutigen Kommunikationsmöglichkeiten wie E-Mail oder Video Konferenz umgangen werden können. Weitere Informationen sind in vielen Publikationen veröffentlicht, siehe zum Beispiel [3].

IXP Industrial eXtreme Programming ist ebenfalls aus den Erfahrungen von XP entstanden und wird häufig im industriellen Kontext verwendet. Bei IXP werden neue Werte wie Vergnügen und Lernen beschrieben und es kommen interessante Managementtechniken hinzu, die in XP so noch nicht vorhanden sind. Eine nähere Erläuterung ist darüber im Buch [4] zu finden.

2.6 Vergleich zu Scrum

Wie XP zählt auch Scrum zu einer agilen, iterativen Entwicklungsmethodik. Ein *Sprint* in Scrum umfasst 30 Tage und wird von einem täglichen Treffen (*Daily Scrum Meeting*) begleitet. Wie aus der nachfolgenden Abbildung 2.3 ersichtlich, kann bei Scrum der Kunde nur am Ende eines Sprints Einfluss auf die Entwicklung nehmen (monatlich), XP bietet dem Kunden deutlich mehr Möglichkeiten, in das laufende Projekt einzugreifen (wöchentlich). Bei Scrum wird stark auf die Eigenverantwortlichkeit des Teams gezählt, da es das Erreichen der Ziele im Sprint intern organisiert.
Ein Nachteil von XP ist, dass die Techniken teils aufwändig oder gewöhnungsbedürftig sind und vom Team zunächst eingeübt werden müssen. Somit ist die Einführungsphase mit mehr Aufwand verbunden und kann schon mal ein halbes Jahr benötigen. Anschließend ist es aber möglich mit XP durchaus schneller Ergebnisse zu erzielen, als mit anderen Entwicklungsmethoden, weil XP-Techniken sehr effektive Vorgehensweisen beschreiben. Scrum benötigt diese Einarbeitungszeit nicht und kann bei XP-unerfahrenen Teams und kleinen Projekten unter Umständen oft schneller zur Fertigstellung führen.

Im Allgemeinen ist es sinnvoll Scrum einzusetzen, wenn XP-Techniken nicht anwendbar sind.

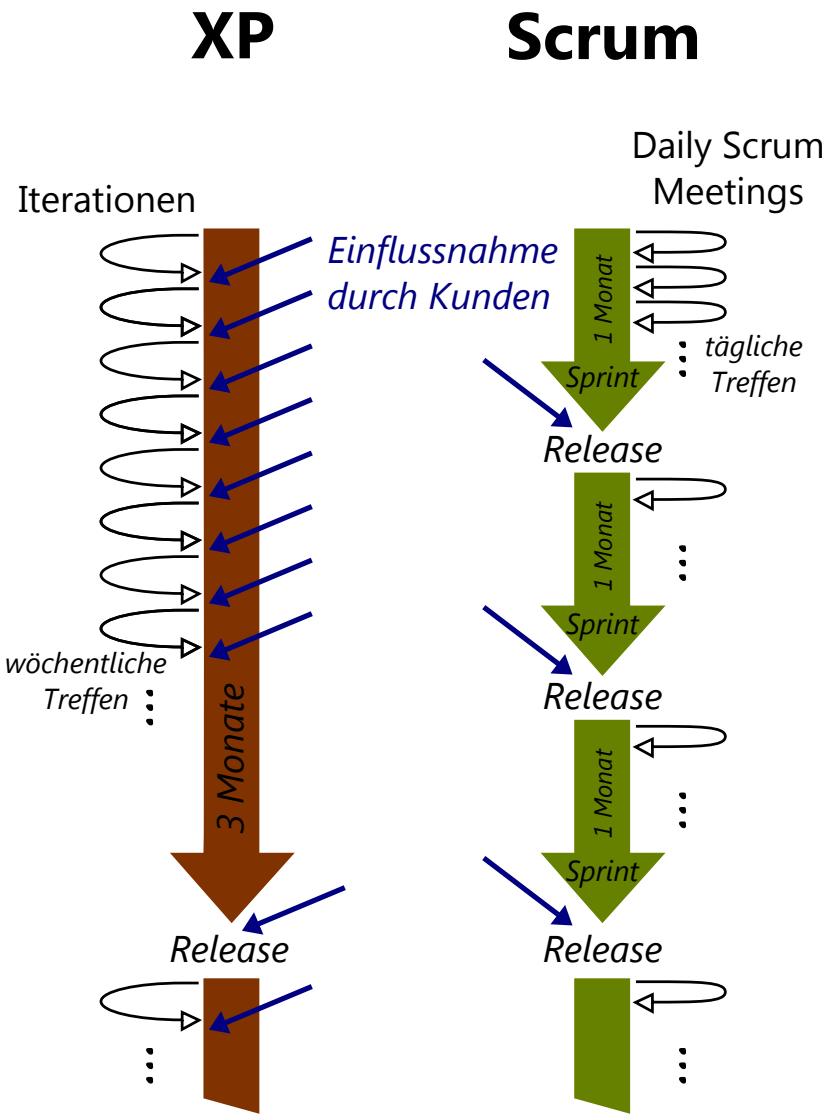

Abbildung 2.3: Iterationen von XP und Scrum im Vergleich

Kapitel 3

Zusammenfassung und Ausblick

Man kann behaupten, dass XP ein Meilenstein der agilen Softwareentwicklung darstellt. Seit der Veröffentlichung von Becks erstem Werk wurden weltweit viele Projekte mit XP erfolgreich umgesetzt. Dabei ist es womöglich nicht immer leicht, allen Prinzipien Folge zu leisten. Doch kann eine disziplinierte Anwendung durchaus zu effektiveren und attraktiveren Lösungen führen, als andere Entwicklungsmethoden. Durch seine wöchentlichen Iterationen ist XP sehr flexibel und kann auf sich ändernde Anforderungen einfach, schnell und effektiv reagieren.

Nicht nur neu zusammengestellte, sondern auch bereits gut interagierende Teams können an XP Gefallen finden und neue Herausforderungen bewältigen. XP wird wahrscheinlich noch für lange Zeit eine gute Entwicklungsgrundlage bieten und in den nächsten Jahren von vielen angewendet und erweitert werden.

Literaturverzeichnis

[1] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change*. The XP Series. Addison-Wesley, 2005.

[2] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[3] Michael Kircher, Prashant Jain, Angelo Corsaro, and David Levine. Distributed extreme programming. In *Second international conference on eXtreme Programming and Agile Processes in Software Engineering*, pages 66–71, 2001.

[4] H. Wolf, S. Roock, and M. Lippert. *Extreme programming: eine Einführung mit Empfehlungen und Erfahrungen aus der Praxis*. dpunkt-Verl., 2005.