

1 Einleitung

- Kann man Software nicht wesentlich schneller entwickeln, als es heute üblich ist?
- Führen die gängigen Entwicklungsprozesse sicher zu hochqualitativer Software?
- Ist das Risiko für gescheiterte Softwareprojekte so gering, wie das in anderen Entwicklungsdisziplinen der Fall ist?

Diese und ähnliche Fragen werden seit über fünf Jahren heiß diskutiert. Ein wesentlicher Auslöser ist die Diskussion um *eXtreme Programming (XP)*. Wir können diese Fragen natürlich auch nicht endgültig beantworten, wollen aber auf Grundlage unserer Erfahrungen mit XP einige Denkanstöße geben. Wir hoffen, dass wir in diesem Buch unsere Erfahrungen so aufbereitet haben, dass sie auch beim Ausprobieren hilfreich sind. Letztlich kann sich jedes Entwicklungsteam die oben aufgeführten Fragen nur selbst auf Basis eigener Erfahrung beantworten.

XP ist eine relativ neue Entwicklungsmethodik, die in den letzten fünf Jahren unglaublich bekannt geworden ist. Dazu haben wesentlich die provokanten Thesen von XP beigetragen. So verwundert es auch nicht weiter, dass XP in einigen Kreisen regelrecht *berüchtigt* ist. Neuerdings reicht es für Entwicklungsmethoden nicht mehr aus, dass sie besonders mächtig und allumfassend sind. Ganz im Gegenteil wird immer häufiger ein schlanker, leichtgewichtiger (agiler) Entwicklungsprozess angestrebt, so dass heute alles ausgiebig gerechtfertigt werden muss, was zusätzlich in eine Entwicklungsmethode aufgenommen wird. In einigen Fällen mag der Puritismus zu weit gehen. Fest steht aber, dass XP viel frischen Wind in die Diskussion um Entwicklungsmethoden gebracht hat.

Kent Beck hat als Vater von XP im Jahr 2000 das erste Buch zu dem Thema veröffentlicht (siehe [Beck 00]) und knapp fünf Jahre später mit der deutlich überarbeiteten Auflage des XP-Manifests (siehe [Beck

04]) die Messlatte noch einmal ein gutes Stück höher gelegt. Wir werden uns im Folgenden häufig auf diese Bücher beziehen. XP ist allerdings nicht ausschließlich von Kent Beck entwickelt worden. Eine Reihe seiner Kollegen, sowohl aus seinem direkten Umfeld als auch aus der Software-Engineering-Gemeinde, setzt ebenfalls XP ein, und inzwischen hat sich eine ganze Community zu dieser Art der Softwareentwicklung zusammengefunden. Prominente Mitglieder sind unter anderem Ron Jeffries, Martin Fowler, Ward Cunningham und Robert C. Martin, die ebenfalls Bücher zu Themen rund um XP und agile Methoden veröffentlicht haben¹. Auf Konferenzen ist das Thema mit entsprechenden Tutorials, Workshops und Panels präsent. Seit dem Jahr 2000 gibt es sogar eine spezielle Konferenz zum Thema eXtreme Programming und agile Entwicklungsprozesse.

Unser Hintergrund

Wir nutzen XP-Techniken seit Anfang 1999 für die Anwendungsentwicklung in einer Reihe von kommerziellen Projekten² (siehe Kapitel 9). Wir wissen, dass XP große Potenziale für die Softwareentwicklung bietet. In diesem Buch wollen wir unsere Erfahrungen mit XP schildern und damit anderen Softwareentwicklern in ihren Projekten helfen, XP erfolgreich anzuwenden. Wir beschäftigen uns sowohl im Rahmen der professionellen Projektarbeit als auch auf den entsprechenden Konferenzen mit XP und tauschen unsere Erfahrungen mit anderen aus der Community aus. Beide Linien sind in dieses Buch eingeflossen.

In den letzten Jahren haben wir mit zu vielen Personen XP-Erfahrungen und -Konzepte diskutiert, als dass wir alle namentlich nennen könnten. Wir möchten daher stellvertretend denen danken, mit denen wir besonders intensive Diskussionen geführt haben:

Dank ■ Mit den Mitgliedern der deutschsprachigen XP-Mailingliste haben wir einen Teil der hier vorgestellten Erfahrungen und Konzepte diskutiert. Die Erfahrungen der Mitglieder der Mailingliste haben uns

1. Die XP-Bücher sind so erfolgreich, dass sie ins Deutsche übersetzt sind oder die Übersetzung für die nahe Zukunft ansteht.

2. Wir werden manchmal gefragt, wie wir seit Anfang 1999 XP-Techniken einsetzen können, wenn das XP-Buch von Kent Beck erst Ende 1999 erschienen ist. Die Techniken sind allerdings nicht mit dem Buch von Kent Beck zum ersten Mal beschrieben und diskutiert worden. Viele der Techniken wurden schon Jahre vorher in unterschiedlichen Kontexten erörtert. So finden sich viele XP-Ansätze inkl. dem Programmieren in Paaren (engl. *Pair Programming*) in [Cunningham 96]. Zudem haben wir bereits im April 1999 an einem Workshop von Kent Beck zum Thema »eXtreme Programming« teilgenommen.

immer wieder neue Perspektiven und Zugangsweisen zu XP eröffnet.

- In ganz Deutschland existieren XP-User-Groups, die sich mehr oder weniger regelmäßig treffen. Die XP-User-Group-Hamburg trifft sich ca. einmal im Monat. Auch hier haben sich wertvolle Diskussionen ergeben.
- Mit Jutta Eckstein, Johannes Link, Frank Westphal, Tammo Freese, Christian Wege und Hans Wegener haben wir interessante Gespräche auf unterschiedlichen Konferenzen geführt.
- Auf den XP-Konferenzen haben wir eine Reihe von Artikeln veröffentlicht und sowohl von den Reviewern als auch von den Teilnehmern der Konferenzen wichtiges Feedback erhalten.
- Jürgen Ahting, Alex Beppe, Jutta Eckstein, Tammo Freese, Dierk König, Johannes Link, Matthias Lübken, Helmut Neukirchen, Björn Ostermann, Marko Schulz und Frank Westphal haben frühe Versionen dieses Buches gelesen und uns detailliertes und konstruktives Feedback gegeben.
- Bernd Oestereich hat die Beschreibung des V-Modells XT in Kapitel 3 beigesteuert.
- Olaf Thiel und Sebastian Sanitz waren bereit, sich für Kapitel 2 beim Programmieren in Paaren ablichten zu lassen.
- Jürgen Ahting hat für das Kapitel 6 den Abschnitt 6.7 beigesteuert, der die Sicht des Kunden auf Softwareprojekte beleuchtet.
- 2000 bis 2004 arbeiteten wir als Berater und Entwickler bei der C1 Workplace Solutions GmbH. Die Mitarbeiter standen uns immer wieder als Gesprächspartner für unsere zum Teil recht »spinnernten« Ideen zur Verfügung; unsere damaligen Chefs haben uns gewähren lassen.
- Seit 2005 arbeiten wir im Rahmen der it-agile GmbH fokussiert an der Anwendung und Weiterentwicklung agiler Methoden. Den Kollegen der it-agile gebührt unser Dank für ihre Unterstützung.
- Im Rahmen unserer Arbeit haben wir eine Vielzahl von Entwicklungsprojekten begleitet und selbst durchgeführt. In diesen Projekten haben wir die Erfahrungen machen können, die wir hier vorstellen. Allen Projektpartnern gebührt Dank für die fruchtbare Zusammenarbeit.
- Neben der Arbeit in kommerziellen Projekten arbeiteten wir als wissenschaftliche Mitarbeiter am Fachbereich Informatik der Universität Hamburg. Die Mitarbeiter am Arbeitsbereich Softwaretechnik haben unsere Arbeit nach Kräften unterstützt und uns die Möglichkeit gegeben, in Seminaren und Praktika XP zu thematisieren.

- Außerdem möchten wir dem dpunkt.verlag und persönlich Christa Preisendanz für die Betreuung dieses Buchprojektes danken.
- Nicht zuletzt gehört unser Dank unseren Frauen für mehr, als man hier hinschreiben könnte.

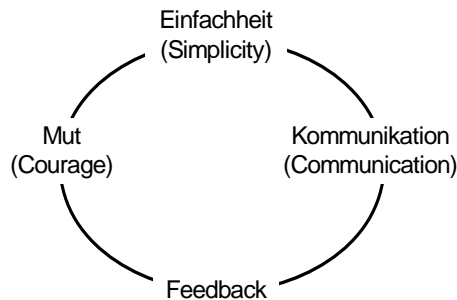
Terminologie XP stammt aus dem englischen Sprachraum. Wir haben deutsche Übersetzungen verwendet, wo diese in der XP-Szene akzeptiert sind, geben aber jeweils den englischen Originalbegriff mit an. Ist keine akzeptierte Übersetzung vorhanden, verwenden wir den englischen Originalbegriff.

Überblick über dieses Kapitel Im Rest dieses Kapitels geben wir eine Kurzübersicht über XP. Dazu beschreiben wir zuerst die XP-Werte, kommen dann zu den Prinzipien und schließlich zu den XP-Techniken. Schließlich veranschaulichen wir den Unterschied zwischen klassischen Vorgehensweisen und XP anhand eines Beispielprojektes. Wir beschreiben zuerst, wie ein klassisches Projekt verlaufen könnte, und anschließend, wie ein XP-Projekt ablaufen könnte. Dabei haben wir die Unterschiede auf die Spitze getrieben, um sie zu kontrastieren.

1.1 Die XP-Werte

Vier Werte XP ruht auf vier Werten: Einfachheit (engl. *Simplicity*), Kommunikation (engl. *Communication*), Feedback (engl. *Feedback*) und Mut (engl. *Courage*) (siehe Abb. 1–1). In diesen Werten liegt der wesentliche Schlüssel zu XP. Wer diese Werte lebt, kann die XP-Techniken einfach erlernen und erfolgreich einsetzen. Wer sich mit den Werten nicht identifizieren kann, dem werden die XP-Techniken auch nicht wesentlich weiterhelfen.

Abb. 1–1:
XP-Werte



XP will möglichst *einfache Lösungen* finden, weil sie schneller und kostengünstiger zu realisieren sind als komplexe Lösungen. Außerdem lassen sich einfache Lösungen leichter erklären sowie einfacher warten und weiterentwickeln. Aus dieser Erkenntnis entstammt die Leitlinie, stets die einfachste Lösung zu realisieren (engl. *The simplest thing that could possibly work*). Einfachheit bezieht sich im Übrigen auch auf den Entwicklungsprozess. Daher ist XP selbst einfach gehalten.

Einfache Lösungen

Ein wichtiger Punkt bei jedem Softwareentwicklungsprojekt ist die Qualitätssicherung. In XP-Projekten wird Qualität über *Feedback* gesichert. Das Feedback findet dabei auf verschiedenen Ebenen statt. Entwickler schreiben Komponententests, um Feedback über die Korrektheit ihrer Software zu erhalten. Neue Systemversionen werden den Anwendern möglichst schnell zur Verfügung gestellt, um von den Anwendern Feedback darüber zu erhalten, ob die Software bei der Aufgabenerledigung hilft.

Feedback

Alle Projektmitglieder sollen intensiv miteinander *kommunizieren*. Dabei steht das persönliche Gespräch im Vordergrund, da auf diesem Wege Informationen am effektivsten auszutauschen sind. Insbesondere können Missverständnisse und Unklarheiten direkt ausgeräumt werden. Wenn eine intensive Kommunikation der Projektmitglieder sichergestellt ist, kann auf einen Gutteil der sonst üblichen Dokumentation verzichtet werden. Eine geschriebene Dokumentation kann aus unterschiedlichen Gründen weiterhin sinnvoll oder notwendig sein (z. B. Revisionsicherheit in Banken und Versicherungen). Allerdings ist diese Dokumentation für den Projekterfolg nicht primär entscheidend.

Kommunikation

Einfache Lösungen, Feedback und Kommunikation erfordern eine Menge *Mut*. Dies gilt vor allem dann, wenn man es nicht gewohnt ist, nach diesen Werten zu handeln. Einfache Lösungen erfordern Mut, weil man etwas weglassen muss. Man muss die Aspekte bewusst *abwählen*, die weniger wichtig sind und die Lösung komplex machen würden. In diesem Sinne braucht man für einfache Lösungen den *Mut zur Lücke*. Feedback erfordert Mut, weil eine negative Bewertung als persönliche Kritik aufgefasst werden könnte. Entwickler halten sich manchmal – bewusst oder unbewusst – für unfehlbar und können Kritik an der von ihnen entwickelten Software nur schwer ertragen. Kommunikation erfordert Mut, weil sich herausstellen kann, dass man einem Missverständnis aufgesessen ist und etwas an der erstellten Software wieder ändern muss. Bei der Kommunikation mit Kunden und Anwendern müssen sich die Entwickler auf deren Fachsprache einlassen. Sie entfernen sich damit von dem festen Boden der ihnen bekannten Technologien und müssen Unwissenheit im Anwendungsbereich

Mut

eingestehen. Schließlich müssen Entwickler mit ihren Vorgesetzten kommunizieren, um XP in einem Unternehmen einzuführen.

Sprechen wir von einem XP-Projekt oder von eXtreme Programming allgemein, so sprechen wir immer über die beteiligten Menschen. Durch die Werte wird deutlich, dass die Projektbeteiligten im Mittelpunkt stehen, im Gegensatz zu Werkzeugen, Prozessen oder dem Produkt.

1.2 Die XP-Prinzipien

XP-Prinzipien Die 15 XP-Prinzipien leiten sich aus den XP-Werten ab. Die fünf zentralen Prinzipien sind

- *Unmittelbares Feedback* (engl. *Rapid Feedback*)
- *Einfachheit anstreben* (engl. *Assume Simplicity*)
- *Inkrementelle Veränderung* (engl. *Incremental Change*)
- *Veränderung wollen* (engl. *Embracing Change*)
- *Qualitätsarbeit* (engl. *Quality Work*).

Unmittelbares Feedback *Unmittelbares Feedback*: Feedback sollte so schnell wie möglich eingeholt werden. Aus der Lernpsychologie weiß man, dass der zeitliche Abstand zwischen Aktion und Feedback eine entscheidende Rolle für den Lernerfolg spielt. Je kürzer die zeitliche Differenz, desto größer der Lernerfolg. Gleichzeitig verhindern wir durch unmittelbares Feedback, dass wir unnötig lange in die falsche Richtung denken und arbeiten.

Einfachheit anstreben *Einfachheit anstreben*: Einfache Lösungen haben eine Reihe wichtiger Vorteile gegenüber komplizierten Lösungen. Sie sind schneller zu erstellen, so dass wir schneller Feedback bekommen können. Einfache Lösungen sind leichter zu verstehen und damit leichter zu kommunizieren. Nicht zuletzt sind einfache Lösungen schneller änderbar, wodurch wir im Entwicklungsprozess flexibler auf geänderte Gegebenheiten reagieren können. Das Streben nach Einfachheit bereitet vielen Softwareentwicklern zunächst Probleme, da uns (nicht nur) in unserer Ausbildung immer wieder eingebläut wurde, dass wir für die Zukunft planen müssen. Es mag paradox erscheinen, aber gerade durch einfache Lösungen sind wir bestens für die Zukunft vorbereitet. Da wir schlichtweg nicht wissen, welche Anforderungen die Zukunft für uns bereithält, ist es am effektivsten, nicht über zukünftige Anforderungen zu spekulieren. Stattdessen sind wir darauf vorbereitet, jede beliebige in der Zukunft auftretende Anforderung umzusetzen, weil unsere einfachen Lösungen leicht änderbar sind.

Inkrementelle Veränderung *Inkrementelle Veränderung*: Auch wenn wir Einfachheit anstreben, werden Softwaresysteme im Laufe ihrer Entwicklung so komplex,

dass Änderungen unerwartete Effekte verursachen können. Nur wenn wir stets kleine Änderungen vornehmen, bleiben diese Effekte beherrschbar. Jeder so auftretende Seiteneffekt kann leicht in Beziehung zu kleinen Änderungen gesetzt werden. Nehmen wir hingegen große Veränderungen vor, so bedeutet dies ein sehr hohes Risiko, dass nicht beherrschbare Seiteneffekte auftreten. Dann ruft nämlich eine Menge von Änderungen eine Menge von Seiteneffekten hervor und wir können nicht feststellen, in welcher Beziehung diese zueinander stehen.

Veränderung wollen: Fortschritt bedeutet Veränderung. Nur, wer Veränderungen will, wird mit Fortschritt belohnt. Veränderung bedeutet aber auch, dass man seine *Komfortzone* verlassen muss. Man muss neue Dinge wagen und Fehlschläge riskieren – immer in dem Bewusstsein, dass man vor allem aus Fehlern lernt.

Veränderung wollen

Qualitätsarbeit: Kein Softwareentwickler programmiert gerne schlechte Software. Softwareentwickler liefern gerne Qualitätsarbeit. Das erhöht ihre Arbeitszufriedenheit und damit wiederum ihre Produktivität. In einem XP-Projekt muss daher die Möglichkeit geschaffen werden, Qualitätsarbeit abzuliefern. Allerdings muss jeweils klar definiert sein, wer bestimmt, ob eine hohe oder niedrige Qualität vorliegt. Mitunter verwechseln Softwareentwickler ihre eigenen Qualitätsmaßstäbe mit denen der Anwender. Letztlich geht es darum, für die Anwender hochqualitative Software zu entwickeln. Also definieren die Anwender zumindest den Maßstab für die Gebrauchsqualität der Software. Unmittelbares Feedback durch die Anwender hilft, möglichst häufig die Gebrauchsqualität bewerten zu lassen und Erfolgserlebnisse zu produzieren.

Qualitätsarbeit

Die weiteren XP-Prinzipien sind

- *Lernen lehren* (engl. *Teach Learning*)
- *Geringe Anfangsinvestition* (engl. *Small Initial Investment*)
- *Auf Sieg spielen* (engl. *Play to win*)
- *Gezielte Experimente* (engl. *Concrete Experiments*)
- *Offene, aufrichtige Kommunikation* (engl. *Open, honest Communication*)
- *Die Instinkte des Teams nutzen, nicht dagegen arbeiten* (engl. *Work with people's instincts, not against them*)
- *Verantwortung übernehmen* (engl. *Accepted Responsibility*)
- *An örtliche Gegebenheiten anpassen* (engl. *Local Adaptations*)
- *Mit leichtem Gepäck reisen* (engl. *Travel light*)
- *Ehrliches Messen* (engl. *Honest Measurement*).

Lernen lehren: XP wird von einigen Verfechtern dogmatisch vertreten. Allerdings hat XP genau den entgegengesetzten Anspruch. XP will

Lernen lehren

auch lehren zu lernen. Entwickler sollen z. B. nicht den »Befehl« erhalten, eine bestimmte Anzahl von Tests zu schreiben. Stattdessen sollen die Entwickler selbst lernen, wie viele Tests sie benötigen.

*Geringe
Anfangsinvestitionen*

Geringe Anfangsinvestitionen: Wir leben in einer unsicheren Welt, insbesondere was Softwareentwicklung angeht. Auch wenn ein Projekt über einen längeren Zeitraum geplant wird, kann es sein, dass es vorzeitig wieder beendet wird. Um den finanziellen Verlust in solchen Fällen gering zu halten, sollen die Anfangsinvestitionen möglichst klein sein. Gleichzeitig werden Entwickler und Anwender gezwungen, sich in einem engen finanziellen Rahmen zu bewegen und sich auf die wirklich wichtigen Aspekte des Projektes zu konzentrieren. In der Vergangenheit wurden einige Projekte mit sehr großen Budgets und finanziellen Freiheiten durchgeführt. Die meisten endeten in einem Desaster.

Auf Sieg spielen

Auf Sieg spielen: XP-Projekte spielen, um zu gewinnen. Viele andere Projekte spielen, um nicht zu verlieren. Das ist keineswegs das Gleiche. Während das erste Projektteam offensiv die anstehenden Aufgaben angeht, ist das zweite Projektteam stets darauf bedacht, nur keine Fehler zu machen. Das erste Team ist dabei klar im Vorteil. Sie werden natürlich Fehler machen. Sie können jedoch aus den Fehlern lernen und so an ihren Aufgaben wachsen. Die zweite Interpretation dieses Prinzips läuft darauf hinaus, keine Ressourcen in toten Projekten zu verschwenden. Wenn nicht mehr gewonnen werden kann, sollten wir uns dies klar eingestehen und das Projekt lieber beenden.

Gezielte Experimente

Gezielte Experimente: Jedes Mal, wenn wir eine Entscheidung fällen, könnte diese falsch sein. Um Risiken zu minimieren, versuchen wir, möglichst schnell darüber Klarheit zu erlangen, ob eine Entscheidung falsch war. Gezielte Experimente können häufig diese Klarheit bringen. Die Experimente können sich sowohl auf das Softwaresystem als auch auf den Entwicklungsprozess beziehen. Im Softwaresystem nutzen wir Tests, um festzustellen, ob eine Entwurfsentscheidung geeignet ist. Im Entwicklungsprozess führen die Projektmitglieder häufig Reviews über den Prozess durch, um zu beurteilen, ob eine Entscheidung richtig oder falsch war.

*Offene, aufrichtige
Kommunikation*

Offene, aufrichtige Kommunikation: Dieses Prinzip ist im Grunde eine Binsenweisheit. Natürlich können Softwareentwicklungsprojekte nur dann funktionieren, wenn die Projektmitglieder offen und aufrichtig sind. Leider ist dies in vielen Entwicklungsprojekten aufgrund von Ängsten, Eitelkeiten und Rivalitäten nicht gegeben. Diese Defizite in der Kommunikation sind häufig verantwortlich für das Scheitern von Projekten. Damit wird das Herstellen einer offenen und aufrichtigen Kommunikation zu einer zentralen und sehr anspruchsvollen Aufgabe.

Die Instinkte des Teams nutzen, nicht dagegen arbeiten: Einige der XP-Techniken (z. B. Programmieren in Paaren, engl. *Pair Programming*) scheinen ungewöhnlich. Andererseits finden sich Entwickler seit jeher in Paaren zusammen, wenn sie eine besonders schwierige Funktionalität realisieren müssen. Das Programmieren in Paaren kann also als eine instinktive Technik von Softwareentwicklern verstanden werden. Dies gilt für die anderen XP-Techniken in ähnlicher Form. In der Praxis ist dieses Prinzip allerdings nicht einfach umzusetzen, da »Instinkt« und »angelerntes Verhalten« nur schwer zu trennen sind. So kann man in unerfahrenen XP-Teams gelegentlich beobachten, dass das Programmieren in Paaren unter Zeitdruck ebenso aufgegeben wird wie das Schreiben von Tests. Ist dies ein instinktives Verhalten oder fallen die Entwickler zurück in angelernte Verhaltensweisen? Wenn das ganze Team jedoch gemeinsam der Meinung ist, dass etwas Unsinn ist, kann dies ein starkes Indiz dafür sein, dass es tatsächlich Unsinn ist. Diese Erkenntnis findet sich im folgenden Ausspruch wieder: »Was sich wie Quatsch anhört, ist häufig auch Quatsch.«

*Die Instinkte des Teams nutzen,
nicht dagegen arbeiten*

Verantwortung übernehmen: Verantwortung soll nicht zugewiesen, sondern übernommen werden. Wird einem Team die Verantwortung zugewiesen, ein unmögliches Projekt durchzuführen, wird das mit Sicherheit katastrophale Auswirkungen auf die Motivation des Teams und damit auch auf die Projektergebnisse haben. Hat das Team hingegen die Möglichkeit, Verantwortung zu *übernehmen*, so fühlen sich die Entwickler auch verantwortlich für ihre Aufgaben. Das steigert die Motivation im Team und damit die Erfolgchancen des Projektes.

Verantwortung übernehmen

An örtliche Gegebenheiten anpassen: Es ist höchst unwahrscheinlich, dass sich XP in genau der von Kent Beck beschriebenen Form in Ihrem Projekt einsetzen lässt. XP muss wahrscheinlich auf die lokalen Gegebenheiten angepasst werden. Schließlich ist das primäre Ziel nicht die exakte Nachahmung, sondern die erfolgreiche Projektdurchführung. Jede Anpassung von XP, die dieses Ziel unterstützt, ist gerechtfertigt.

An örtliche Gegebenheiten anpassen

Mit leichtem Gepäck reisen: Mit schwerem Gepäck kann man sich nicht schnell vorwärts bewegen. Daher sollte das »Marschgepäck« sorgfältig ausgewählt werden. Es sollte insbesondere aus wenigen einfachen, aber sehr nützlichen Dingen bestehen (z. B. Wiki-Web als Wissensbasis im Projekt). Dabei spielt die flexible Kombinierbarkeit dieser Dinge eine wichtige Rolle.

Mit leichtem Gepäck reisen

Ehrliches Messen: Um den Entwicklungsprozess unter Kontrolle zu haben, müssen Messungen unterschiedlichster Art vorgenommen werden. So messen wir mit Komponententests z. B. das korrekte Funk-

Ehrliches Messen

tionieren unserer Klassen. Alle diese Messungen haben jedoch nur dann einen Sinn, wenn das Team ehrlich misst. So sehen erfolgreiche Tests zwar sehr schön aus. Wenn sie aber nur deshalb erfolgreich sind, weil die Entwickler alle nicht funktionierenden Teile auskommentiert haben, ist die Messung nicht mehr aussagekräftig. Sie kann sogar negative Auswirkungen auf das Projekt haben, weil sie eine Sicherheit suggeriert, die so nicht existiert.

1.3 Die XP-Techniken

Die vier XP-Werte und die 15 XP-Prinzipien werden durch 14 XP-Techniken³ unterstützt. Sie helfen den Entwicklern, sich »prinzipientreu« zu verhalten. Die XP-Techniken sind im Überblick in Abbildung 1–2 dargestellt. Sie werden in den folgenden Kapiteln detailliert beschrieben.

Begriff »Technik«

Im englischen Original heißen die Techniken »Practices«. Da im Deutschen der Begriff »Praktik« etwas anders belegt ist, haben wir uns dafür entschieden, den Begriff »Technik« zu verwenden. Damit geht leider eine Konnotation verloren: »Practice« wie in »Best Practice« bezieht sich immer auch auf Erfahrungen, was beim deutschen Begriff »Technik« nicht automatisch der Fall ist.

3. In ursprünglichen XP-Manifest von Kent Beck sind zwölf XP-Techniken beschrieben. Inzwischen haben sich Standup-Meetings und Retrospektiven als zusätzliche Techniken etabliert.

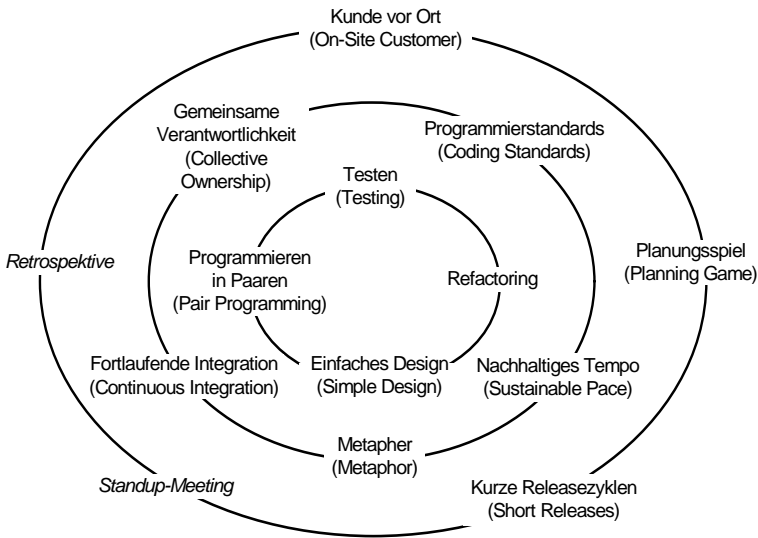
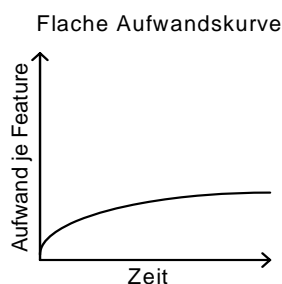
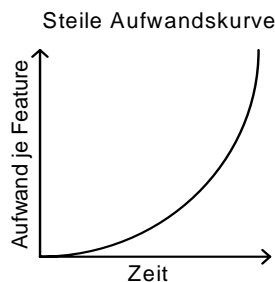


Abb. 1-2:
 XP-Techniken im
 Überblick, angelehnt an
www.xprogramming.com

Eine wichtige Voraussetzung für die Wirksamkeit der XP-Techniken ist eine flache Aufwandskurve (siehe Abb. 1–3). Traditionelle Entwicklungsprozesse gehen häufig von einer steilen Aufwandskurve aus. Demnach wird die Realisierung einer Anforderung mit der Zeit exponentiell teurer, wenn die Anforderung erst spät entdeckt wird. Also ist die vollständige Analyse aller Anforderungen zu Projektbeginn eine unabdingbare Notwendigkeit. XP hingegen geht davon aus, dass der Aufwand für die Umsetzung einer Anforderung im Wesentlichen unabhängig vom Zeitpunkt der Umsetzung ist. Insbesondere geht XP davon aus, dass die Aufwandskurve über die *Zeit nicht* exponentiell wächst. Damit entfällt der Zwang, alle Anforderungen bereits zu Projektbeginn zu kennen. Aus diesem Grund konzentriert sich XP stets auf die *aktuellen* Aufgaben.

Der Verlauf der Aufwandskurve hängt wesentlich von der verwendeten Technologie ab. So führen die traditionellen imperativen Programmiersprachen häufig zu steilen Aufwandskurven, während objektorientierte Technologien zumindest das Potenzial für flache Aufwandskurven mitbringen. Ob diese flache Aufwandskurve in Projekten tatsächlich erreicht werden kann, hängt wesentlich von der Verwendung der eingesetzten Technologien ab. Insbesondere sollte das DRY-Prinzip (*DRY: Don't repeat yourself*) beachtet werden, wonach jede Kernidee im System an genau einer Stelle ausgedrückt werden soll.

Abb. 1-3:
Steile vs. flache
Aufwandskurve



Ist die Aufwandskurve nicht flach, können viele XP-Techniken nicht umgesetzt werden (z. B. *Refactoring*, *einfaches Design*, *kurze Releasezyklen*, *fortlaufende Integration*). Nach unserer Erfahrung sind objektorientierte Technologien geeignet, um eine flache Aufwandskurve zu erreichen. Sie geben jedoch keine Garantie. Wir müssen die Technologien auch geeignet anwenden. Insbesondere sollten Kernabstraktionen der Anwendung in überschaubar wenigen Systemkomponenten umgesetzt werden. Zusätzlich spielen die verfügbaren Tools eine große Rolle. Java mit Refactoring-Browser à la Eclipse lässt sich viel flexibler handhaben als C++ ohne Refactoring-Unterstützung.

1.3.1 Managementtechniken

Kunde vor Ort XP-Projekte sind stets an den fachlichen Anforderungen orientiert und beziehen Anwender und Kunden⁴ ins Projekt mit ein (*Kunde vor Ort*, engl. *On-Site Customer*, siehe Abschnitt 2.1). Anwender und Kunden geben die fachlichen Anforderungen für das Projekt vor. Entwickler müssen diese Anforderungen *verstehen* und bekommen von Anwen-

4. In XP wird keine Differenzierung zwischen Anwendern und Kunden vorgenommen. Wir werden später argumentieren, dass eine solche Differenzierung in vielen Fällen notwendig und auch mit XP kompatibel ist.

den und Kunden *keine* Spezifikation des zu erstellenden Systems⁵. Anwender und Kunden stehen den Entwicklern dafür als Ansprechpartner für fachliches Wissen zur Verfügung und können jederzeit befragt werden.

XP-Projekte verlaufen iterativ und inkrementell. Der Umfang des jeweils nächsten Inkrements wird im *Planungsspiel* (engl. *Planning Game*, siehe Abschnitt 2.2) zusammen von Anwendern, Kunden und Entwicklern festgelegt. Dabei priorisieren die Anwender und Kunden die zu realisierenden Anforderungen. Die Entwickler sind für die Aufwandsschätzung der einzelnen Anforderungen zuständig. Die Priorisierung der Anforderungen kann von den geschätzten Aufwänden abhängen.

Planungsspiel

Alle Projektteilnehmer treffen sich täglich zu einer festgelegten Zeit zum *Standup-Meeting* (siehe Abschnitt 2.3), um sich für 15 Minuten über den Projektfortschritt auszutauschen.

Standup-Meetings

Neue und geänderte Systemkomponenten sollen in kurzen Abständen (*kurze Releasezyklen*, engl. *Short Releases*, siehe Abschnitt 2.5) den Anwendern zur Verfügung gestellt werden. Dadurch können die Anwender und Kunden schnell vom System profitieren. Außerdem können die freigegebenen Versionen schnell durch die Anwender bewertet werden. Diese Bewertung kann dann zusammen mit ggf. geänderten Anforderungen in die weitere Entwicklung eingehen.

Kurze Releasezyklen

Retrospektiven (engl. *Retrospective*, siehe Abschnitt 2.6) in der Softwareentwicklung sind rückblickende Reflexionen der Projektbeteiligten. Mit Hilfe von Retrospektiven sollen Probleme und Blockaden im Softwareentwicklungsprozess gefunden und für die zukünftige Entwicklung beseitigt werden. Retrospektiven werden in der Regel in Abständen von ein bis sechs Monaten mit allen oder ausgewählten Projektbeteiligten durchgeführt. Eine Retrospektive dauert zwischen einem halben und drei Tagen.

Retrospektive

1.3.2 Teamtechniken

XP-Systeme sollen wenigen klaren *Metaphern* (engl. *Metaphor*, siehe Abschnitt 2.4) folgen, welche die Kernideen des Systems beschreiben. Damit wird den Entwicklern eine Richtlinie für die Systementwicklung gegeben, welche die konsistente Entwicklung des Systems unterstützt.

Metapher

5. »Keine Spezifikation« bezieht sich hier auf herkömmliche Spezifikationsdokumente wie Pflichtenhefte oder Ähnliches, in denen die komplette Systembeschreibung haarklein enthalten sein muss. Im Rahmen der Qualitätssicherung mittels Testens übernehmen Akzeptanztests (siehe *Testen*, Abschnitt 2.7) in gewisser Weise die Rolle der Spezifikation.

Gemeinsame Verantwortlichkeit In XP-Projekten sind alle Entwickler für das System *gemeinsam verantwortlich* (engl. *Collective Ownership*, siehe Abschnitt 2.11). Dies impliziert insbesondere, dass sämtlicher Quellcode von jedem Entwickler zu jedem Zeitpunkt geändert werden darf. Dadurch ist sichergestellt, dass die Entwicklung nicht dadurch ins Stocken gerät, dass einzelne Entwickler nicht verfügbar sind. Ist ein Entwickler nicht verfügbar, kann jeder andere Entwickler seine Arbeit fortführen.

Fortlaufende Integration Änderungen am System werden möglichst schnell integriert. Durch diese *fortlaufende Integration* (engl. *Continuous Integration*, siehe Abschnitt 2.12) werden die geänderten Systemteile den anderen Entwicklern schnell zur Verfügung gestellt. So können Änderungen sofort vom ganzen Team getestet werden. Außerdem werden die Integrationsaufwände insgesamt reduziert, da Konflikte früher erkannt und behoben werden können.

Programmierstandards Damit das *Programmieren in Paaren* und *gemeinsame Verantwortlichkeit* funktionieren können, muss der Quellcode einheitlich gestaltet sein. Dazu werden pragmatische *Programmierstandards* (engl. *Coding Standards*, siehe Abschnitt 2.13) definiert.

Nachhaltiges Tempo In einem XP-Projekt sollen die Entwickler nicht mehr als ihre Regelarbeitszeit arbeiten, also mit einem *nachhaltigen Tempo* (engl. *Sustainable Pace*, siehe Abschnitt 2.14) vorgehen. Dadurch ist es möglich, ihre Kreativität und ihr Engagement über einen langen Zeitraum zu erhalten. Überstunden werden in XP-Projekten nur in engen Grenzen geduldet. Längere Perioden mit Überstunden deuten auf ein Problem hin, beispielsweise ist XP nicht komplett oder adäquat umgesetzt worden.

1.3.3 Programmiertechniken

Testen Alles, was programmiert wurde, muss zur Qualitätssicherung getestet werden (*Testen*, engl. *Testing*, siehe Abschnitt 2.7). Zum einen werden Komponententests programmiert, die automatisiert ausgeführt werden können und die Funktion einzelner Systemkomponenten testen. Zusätzlich wird die Erfüllung fachlicher Anforderungen mit Akzeptanztests geprüft. Der endgültige Akzeptanztest findet allerdings erst durch die Verwendung des Systems durch die Anwender statt.

Einfaches Design Das entwickelte Softwaresystem sollte möglichst einfach gestaltet werden (*einfaches Design*, engl. *Simple Design*, siehe Abschnitt 2.8). Einfache Entwürfe können einfacher und schneller umgesetzt werden. Außerdem sind sie leichter zu verstehen, schneller zu kommunizieren und einfacher zu testen.

Strukturdefizite im Entwurf der entwickelten Software behindern die Weiterentwicklung. Sie werden daher *sofort* behoben. Diese Umstrukturierungen von Software unter Beibehaltung der Funktionalität nennt man *Refactoring* (siehe Abschnitt 2.9). Durch die Komponententests wird sichergestellt, dass das Refactoring keine unerwünschten Seiteneffekte erzeugt.

Refactoring

In XP-Projekten sitzen stets zwei Entwickler vor einem Rechner. Durch dieses *Programmieren in Paaren* (engl. *Pair Programming*, siehe Abschnitt 2.10) wird die Qualität der entwickelten Software erhöht. Außerdem verbreitet sich durch wechselnde Paare schnell das Wissen über das Softwaresystem im Projekt.

Programmieren in Paaren

1.4 Projektbeispiel

Vielfach behaupten Entwickler oder Manager, sie würden im Grunde schon seit Jahren XP anwenden, vielleicht mit geringen Einschränkungen. Kent Beck spricht jedoch davon, dass mit 80 % der XP-Techniken nur 20 % des Effektes erzielt wird. Man kann sicher darüber streiten, ob diese Aussage wirklich zutrifft. Zwischen einem Projekt, in dem einige der beschriebenen Techniken angewendet werden, und einem XP-Projekt besteht jedoch ein himmelweiter Unterschied.

(K)ein XP-Projekt

Im Folgenden skizzieren wir zwei Projektverläufe, die wir in ähnlicher Form bereits mehrfach erlebt haben. Das erste Projekt ist mit Sicherheit *kein* XP-Projekt, da hier keine einzige XP-Technik verwendet wird. Währenddessen nutzt das zweite Projekt die Vorteile von XP. In der ersten Projektbeschreibung haben wir absichtlich fast alle XP-Lehren ins Gegenteil verkehrt, um auf die Effekte der XP-Techniken zu fokussieren. Das soll natürlich keineswegs darüber hinwegtäuschen, dass auch ohne XP erfolgreiche Projekte durchgeführt wurden und immer noch werden.

1.4.1 Kein XP-Projekt

Es soll ein existierendes Softwaresystem ersetzt werden, da dieses nicht mehr den aktuellen Anforderungen genügt. Insbesondere ist das System nicht flexibel genug und Erweiterungen sind insgesamt zu aufwändig. Daher wird beschlossen, das neue System mit objektorientierten Technologien zu realisieren. Als Projektlaufzeit werden 12 Monate anvisiert.

Zuerst wird das Projektteam zusammengestellt. Dazu werden drei Entwickler mit Erfahrungen in objektorientierter Entwicklung ausgewählt. Weiterhin nehmen an dem Projekt drei Entwickler teil, die bis-

her lediglich Schulungen in der Objektorientierung genossen haben. Um die Entwicklung der existierenden Anwendungen nicht zu beeinträchtigen, werden die Entwickler nur mit 50 % ihrer Arbeitszeit dem Projekt zugeteilt.

Um gleich zu Anfang eine ausgereifte Entwicklungsumgebung zur Verfügung zu haben, wird bereits die Technologie ausgewählt. Es wird eine integrierte Entwicklungsumgebung, ein EJB-Application-Server, ein Webserver mit Servlet-Engine sowie ein UML-Tool beschafft und auf den Arbeitsplatzrechnern installiert.

Jetzt geht es an die fachliche Analyse. Diese wird von Analyseexperten durchgeführt, die Anforderungen erheben und als Anwendungsfälle im UML-Tool formulieren. Neben den textuellen Beschreibungen werden die Abläufe als Sequenz- und Aktivitätsdiagramme im UML-Tool beschrieben. Die so modellierten Sachverhalte werden den Anwendern zur Prüfung vorgelegt. Diese bestätigen die Korrektheit der Modellierung.

Anschließend gehen die Entwickler an die objektorientierte Modellierung des Anwendungssystems. Sie verwenden das Tool und die UML-Diagramme wie Klassendiagramme, Objektdiagramme etc. Die Modellierung findet arbeitsteilig und iterativ statt. Das System wird anhand technischer Kriterien zerteilt und dann von den Entwicklern modelliert. So entstehen Modelle für die fachlichen Kernkonzepte, für die Benutzungsoberfläche sowie den Datenbankanschluss. Die Entwickler bewerten ihre Modellierungen dabei gegenseitig und gehen nach jeder Bewertung in eine neue Iteration, in der sie die neuen Erkenntnisse in ihre Modelle einarbeiten. Während der Besprechungen kommt eine Reihe von technischen Fragen (Performance, Ausfallsicherheit etc.) zur Sprache, die jeweils Eingang in die Modellierung finden.

Schließlich werden aus den Diagrammen im UML-Tool Code-rümpfe generiert, die von den Entwicklern ausimplementiert werden. Hierbei bekommt jeder Entwickler einen Teil der Klassen zugewiesen. Es stellt sich schnell heraus, dass sich die modellierten Klassen nicht problemlos implementieren lassen. An einigen Stellen werden daher weitere Klassen entwickelt und an anderen Stellen werden die Beziehungen zwischen Klassen oder die Klassenschnittstellen geändert.

Das Projekt wird nicht termingerecht abgeschlossen. Da das System auch nach 14 Monaten immer noch nicht fertig ist, wird entsprechender Druck auf die Entwickler ausgeübt. Also liefern die Entwickler ein inkonsistentes und stark fehlerbehaftetes System aus. In der Folge verläuft die Entwicklung chaotisch. Benutzer finden Fehler in der Anwendung und stellen fest, dass viele ihrer Anforderungen

nicht angemessen umgesetzt wurden. Dies wird den Entwicklern mitgeteilt. Diese versuchen, die Änderungen möglichst schnell umzusetzen und degenerieren dabei die interne Struktur der Anwendung. Dadurch steigen die Aufwände für zukünftige Änderungen immer weiter und letztlich ist die Situation mit der neuen Software nicht besser als mit dem alten System.

Bewertung des Projektes

Entwickler, die gerade erst angefangen haben, die Objektorientierung (OO) kennen zu lernen, sind anfangs lange nicht so effektiv wie Entwickler mit OO-Projekterfahrung. Es muss also zumindest bei der Projektplanung berücksichtigt werden, dass die Hälfte der Entwickler am Anfang nicht besonders effektiv sein wird. Diese Entwickler werden außerdem die erfahrenen Entwickler durch Rückfragen etc. »abbrem-sen«.

Entwickler nur Teilzeit in Projekten arbeiten zu lassen, ist ebenso verbreitet wie ineffektiv. Der Kontextwechsel zwischen den Projekten bedeutet für die Entwickler einen Mehraufwand. Wir schätzen aus unserer Erfahrung, dass ein Entwickler, der dem Projekt zu 50 % zugeordnet ist, nicht 50 %, sondern lediglich 30 % bis 40 % seiner Maximalleistung erbringen kann. Es ist besser, weniger Entwickler mit 100 % am Projekt teilnehmen zu lassen.

Es ist keine besonders gute Idee, Technologien einzukaufen, bevor auch nur annähernd klar ist, was entwickelt werden soll. Die Technologien erhöhen zunächst die Komplexität im Projekt, und den unerfahrenen Entwicklern ist unklar, wie diese einzusetzen sind.

Softwareentwicklung ist zuerst ein Lernprozess. Auf jeden Fall müssen Entwickler verstehen, wie Anwender arbeiten und welche Anforderungen an die Software daraus erwachsen. Werden Fachanalysierer zwischen Anwender und Entwickler geschaltet, so findet ein »Stille Post«-Spiel statt, das den Lernprozess behindert. Missverständnisse sind dann an der Tagesordnung. Es ist generell keine gute Idee, Anwendern Diagramme aus dem Bereich der Softwareentwicklung vorzulegen. Aus unserer Erfahrung können Anwender aus dem Repertoire der UML-Beschreibungsmittel lediglich Anwendungsfälle verstehen und bewerten. Ihnen Sequenz- oder Aktivitätendiagramme vorzulegen ist in der Regel sinnlos. Sie können diese Diagramme nicht wirklich bewerten. Fordern Softwareentwickler die Anwender auf, diese Diagramme zu bewerten, so werden sie die Diagramme für angemessen befinden, solange die ihnen wichtig erscheinenden Begriffe auftauchen.

Die Kernkonzepte der Anwendung sollten nicht stark arbeitsteilig modelliert werden. Sie müssen nicht nur allen Entwicklern bekannt sein, sondern auch vom gesamten Projektteam getragen werden. Die Aufteilung nach technischen Gesichtspunkten erscheint Softwareentwicklern zunächst nahe liegend, führt aber in der Regel zu aufwändigen Modellierungen. Insbesondere die Modellierung technischer Komponenten wie der Datenbankanschluss ist dann nicht direkt mit den fachlichen Anforderungen verbunden. Folglich versuchen die Entwickler, alle erdenklichen Anforderungen abzudecken. Das führt zu viel zu komplexen Modellen. Dieser Trend zu komplexen Modellen wird noch verstärkt durch die diskutierten technischen Aspekte wie Performance und Ausfallsicherheit. In der Regel haben die Entwickler zu diesem Zeitpunkt noch überhaupt keine Ahnung bzgl. der Anforderungen in diesem Bereich. Und selbst wenn diese Anforderungen einigermaßen klar sind, ist den Entwicklern immer noch unbekannt, wie sich die modellierten Systeme im Einsatz verhalten werden. Folglich erhöhen die Entwickler die Komplexität der Modellierung, um allen möglichen Problemen in diesen Bereichen von vornherein entgegenzutreten. Dies führt in der Summe zu »Technologie auf Vorrat«, was schlicht Geldverschwendung im großen Stil bedeutet.

Die Zuteilung von Klassen zu Entwicklern führt zu einem hohen Truck-Faktor⁶. Fällt einer der Entwickler aus, so ist die Einarbeitung eines anderen Entwicklers in die Klassen sehr aufwändig. Fällt der Entwickler lediglich für einen begrenzten Zeitraum aus (z. B. wegen Krankheit), so werden die Entwickler sich nicht in seinen Code einarbeiten, sondern darum herum arbeiten. Der Truck-Faktor wird noch weiter durch die Tatsache erhöht, dass sich eben nicht alle benötigten Klassen vorher in einem UML-Tool modellieren lassen. Es ist aus unserer Erfahrung *immer* notwendig, während der Programmierung noch Änderungen an der Modellierung vorzunehmen. Damit sind die ehemals vorgenommenen Modellierungen auch für die Dokumentation wenig nützlich, weil sie nicht den aktuellen Zustand des entwickelten Systems widerspiegeln.

Es ist generell schlecht, wenn Analyse, Entwurf und Konstruktion stark voneinander getrennt sind. Diese Aktivitäten müssen stets eng verzahnt werden, weil sich erst bei der Programmierung Lücken oder Unklarheiten der fachlichen Analyse und des Entwurfs zeigen.

6. Der Truck-Faktor beschreibt die Wahrscheinlichkeit, mit der das Projekt scheitert, wenn ein Entwickler von einem Truck überfahren wird oder aus anderen Gründen nicht mehr für das Projekt zur Verfügung steht (siehe Abschnitt 2.11).

1.4.2 Ein XP-Projekt

Das gleiche Projekt würde als XP-Projekt ganz anders verlaufen.

Zunächst werden die Entwickler zu 100 % dem Projekt zugeordnet. Das Projektteam wird mit offenen Karten bzgl. der Qualifikation der Entwickler spielen. Damit ist den erfahrenen Entwicklern auch klar, dass sie die weniger erfahrenen Entwickler teilweise im Projekt mit ausbilden müssen.

Zunächst kümmert sich das Projekt um eine erste fachliche Analyse. Diese wird im direkten Kontakt mit den Anwendern durchgeführt. Es ist den Entwicklern klar, dass alle jetzt erstellten Dokumente und Diagramme nur zum Lernen dienen und anschließend im Grunde weggeworfen werden können. Daher machen sich die Entwickler ihre Notizen auf Papier oder auf Whiteboards. Wird ein Whiteboard-Bild für so wichtig erachtet, dass es konserviert werden soll, wird es mit einer Digitalkamera abfotografiert oder mit einem passenden Grafikprogramm (z. B. Visio) abgezeichnet.

Sobald die Entwickler einen ersten Überblick haben, beginnen sie sofort mit der Programmierung. Zunächst haben sie noch ein unscharfes Bild über das zu entwickelnde System. Sie erstellen daher einige Prototypen, um noch offene Fragen bzgl. der Handhabung des Systems sowie technischer Aspekte zu klären. Die Handhabungsprototypen werden den Anwendern präsentiert und mit ihnen diskutiert. Sie stellen ein wichtiges Instrument für Anwender und Entwickler dar, um ein gemeinsames Verständnis über das zu entwickelnde System herauszubilden.

Nachdem klar ist, welche Systemteile zuerst entwickelt werden sollen, erstellen Entwickler zusammen mit dem Kunden und den Anwendern eine Planung für die erste Iteration. Für die Iteration wollen sich Anwender, Kunden und Entwickler einen Zeitraum von zwei Wochen nehmen. Die Entwickler schätzen für die einzelnen Anforderungen die Aufwände, und die Kunden und Anwender entscheiden, welche Anforderungen in welcher Reihenfolge umgesetzt werden.

Die Entwickler realisieren anschließend genau den geplanten Funktionsumfang – nicht mehr und nicht weniger. Treten während der Programmierung fachliche Unklarheiten auf, fragen die Entwickler direkt bei den Anwendern nach. Entwickler integrieren Projektfortschritte mehrfach am Tag in den gemeinsamen Bestand. Daher kann zu jedem Zeitpunkt ein vollständig lauffähiges System erstellt werden, wenn dies erforderlich sein sollte. Dann ist es auch keine Katastrophe, dass die Entwickler in den zwei Wochen nicht die gesamte Funktionalität schaffen. Sie liefern nach Ablauf der zwei Wochen den System-

stand, der bis dahin erreicht ist. Auf Basis dieser Erfahrung korrigieren die Entwickler ihr Schätzverfahren, um bei der nächsten Iteration genauere Schätzungen abgeben zu können. Auch wenn sich die Entwickler verschätzt haben, haben die Anwender zum geplanten Zeitpunkt ein einsetzbares System erhalten, das zumindest die wichtigen Funktionalitäten anbietet.

In diesem Verfahren wird so lange weitergearbeitet, bis die gewünschte Gesamtfunktionalität realisiert wurde. Insgesamt kann durch dieses Vorgehen schnell eine erste einsetzbare Systemversion erstellt werden, während gleichzeitig auf neue Anforderungen flexibel reagiert werden kann.

Bewertung des Projektes

Die 100 %-Zuordnung der Entwickler zum Projekt sorgt dafür, dass sich die Entwickler voll und ganz auf das Projekt konzentrieren können. Es entstehen damit keine Reibungsverluste durch den Wechsel zwischen Projekten. Die Offenheit bzgl. der Qualifikationen der Entwickler entspricht dem XP-Prinzip »offene, aufrichtige Kommunikation« und hilft, spätere unangenehme Überraschungen zu vermeiden.

Die fachliche Analyse wird im engen Kontakt mit den Anwendern durchgeführt. Damit wird bereits in diesem frühen Projektstadium akzeptiert, dass die Anwender eine wichtige Rolle bei der Systementwicklung spielen. Die Entwickler halten den engen Kontakt mit den Anwendern während des gesamten Projektverlaufes, weil sie wissen, dass letztlich nicht die Erfüllung eines Pflichtenheftes über den Projekterfolg entscheidet. Sie sind sich darüber im Klaren, dass die Anwender über Projekterfolg oder -misserfolg entscheiden. Nur wenn die Anwender das entwickelte System sinnvoll einsetzen können, wird das Projekt zum Erfolg.

Die Entwickler üben sich über das gesamte Projekt hinweg in »Bescheidenheit«. Insbesondere entwickeln sie nur genau die benötigte Funktionalität und nicht, was »gerade interessant« erscheint. Sie konzentrieren sich in diesem Sinne auf die fachlichen Anforderungen und begreifen Technologie als Mittel zum Zweck. Dadurch können sie ihre Kräfte auf einen Punkt konzentrieren: die fachlichen Anforderungen.

Die kurzen Iterationszyklen sorgen dafür, dass die Anwender stets genaue und verlässliche Informationen über den Projektzustand erhalten. Sie können sich am »lebenden Objekt« vergewissern, was in welcher Qualität realisiert wurde. In diesem Projekt unterscheiden die Entwickler allerdings nicht zwischen Iterationen und Releases, wie dies bei XP üblich ist. In einem größeren XP-Projekt werden in der

Regel mehrere Iterationen zu einem Release (1 bis 3 Monate) zusammengefasst. Ein Release kann im Anwendungsbereich sinnvoll eingesetzt werden, während Iterationen dem schnellen Feedback durch den Kunden und einer verbesserten Planung dienen.

1.5 Überblick über das Buch

Die folgenden Kapitel des Buches gliedern sich in zwei große Bereiche. Kapitel 2 geht ausführlich auf jede einzelne XP-Technik ein. Zu jeder Technik wird neben einer kurzen Beschreibung vor allem auf unsere konkreten Erfahrungen und Empfehlungen fokussiert. Kapitel 3 stellt die bekanntesten anderen agilen Methoden (Scrum, Feature Driven Development, Industrial XP und XP, wie Kent Beck es in der zweiten Auflage seines XP-Buches beschreibt) im Vergleich dar und skizziert auch das neue V-Modell XT. Das V-Modell XT ist die Neuauflage des V-Modell des Bundes. Es soll in der Lage sein, auch agile Projekte zu unterstützen.

Die darauf folgenden Kapitel des Buches (Kapitel 4 bis 9) befassen sich mit speziellen Themen, die sich nicht direkt einzelnen XP-Techniken zuordnen lassen. Wir diskutieren dort zunächst die Rollen, die im Rahmen eines XP-Projektes zu besetzen sind (siehe Kapitel 4). Anschließend gehen wir auf eine Reihe von Artefakten ein, die für ein XP-Projekt verwendet werden können (siehe Kapitel 5). Darauf folgt eine genauere Betrachtung, wie XP-Projekte organisiert werden können und welche organisatorischen Randbedingungen erfüllt werden müssen (siehe Kapitel 6). Kapitel 7 beschreibt die Explorationsphase, mit der XP-Projekte beginnen. Schließlich widmet sich Kapitel 8 der Einführung von XP in Projekten und im Unternehmen. Im Rahmen unserer Erfahrungen arbeiteten wir in sehr unterschiedlichen Projektkontexten. Einige Projekterfahrungen haben wir in Projektberichten in Kapitel 9 zusammengefasst. Im gesamten Buch beziehen wir uns immer wieder auf diese Erfahrungen. Dass XP für unterschiedliche Kontexte jeweils angepasst werden muss, beschreiben wir in Kapitel 10. Hier sind die Kontexte Framework-Entwicklung, Produktentwicklung, Migration von Legacy-Systemen, E-Business, Outsourcing, Zertifizierung und eingebettete Systeme berücksichtigt.

Literaturverweise befinden sich im Text nur an ausgewählten Stellen und sind keinesfalls vollständig. Dafür befindet sich am Ende eines jeden Kapitels eine kommentierte Literaturliste. Dort sind auch zusätzliche, im Text nicht erwähnte, aber aus unserer Sicht interessante Referenzen aufgeführt.

Nach dem Buch

Für die Beschäftigung mit der agilen Softwareentwicklung ist dieses Buch als Einstieg geeignet. Es ist aber bei weitem nicht erschöpfend. Weiterführende Informationen und Diskussionen findet man unter [XPBuch]. Hier kann man auch seine konkreten Fragen diskutieren, die sich vor oder beim Projekteinsatz ergeben.

1.6 Literatur

[AgileAlliance] The Agile Alliance: Agile Manifesto
<http://www.agilealliance.org>

eXtreme Programming gehört zur Gruppe der agilen Prozesse. Neben den Vätern von XP hat sich eine Reihe anderer Interessierter zur Agile Alliance zusammengetan, um die Grundfesten agiler, leichtgewichtiger Softwareentwicklung festzuhalten.

[Beck 00] Kent Beck: eXtreme Programming Explained – Embrace Change, Reading, Massachusetts, Addison-Wesley, 2000.

Das Standardwerk zum Thema eXtreme Programming. In diesem ersten Buch zu XP beschreibt Kent Beck die Grundlagen und generellen Überlegungen zu XP. Das Buch gibt einen guten Überblick über XP und die Philosophie hinter XP, enthält aber wenig Details. Es ist als Einstieg in XP jedoch unverzichtbar. Eine deutsche Übersetzung ist inzwischen von Addison-Wesley veröffentlicht worden.

[Beck 04] Kent Beck: eXtreme Programming Explained – Embrace Change. 2. Auflage, Reading, Massachusetts, Addison-Wesley, 2004.

Die zweite Auflage des XP-Buches ist gegenüber der ersten Auflage vollständig neu geschrieben. Kent Beck hat erhebliche Umbenennungen in der Terminologie und Verschiebungen in der Bedeutung der XP-Techniken vorgenommen. Einige neue Techniken stellen auch die Entwicklungsteams vor neue Herausforderungen, welche die ursprünglichen XP-Techniken beherrschen.

[Beedle & Schwaber 01] Mike Beedle, Ken Schwaber: Agile Software Development with Scrum. Prentice Hall, 2001.

Originalbeschreibung der agilen Methode Scrum mit einer glänzenden ersten Hälfte.

[Cunningham 96] Ward Cunningham: EPISODES: A Pattern Language of Competitive Development. In: Vlissides, Coplien, Kerth

(Eds.): Pattern Languages of Program Design. Addison-Wesley, S. 371-388, 1996.

[ExtremeProgramming] <http://www.extremeprogramming.org>
Website rund um eXtreme Programming (XP).

[Feature Driven Development]
<http://www.featuredrivendevelopment.com>
Website rund um Feature Driven Development (FDD).

[IndustrialXP] <http://www.industrialxp.org>
Einführung in Industrial XP (IXP).

[it-agile] <http://www.it-agile.de>
Website der it-agile GmbH. Hier finden sich im Newsbereich (Blog) Neuigkeiten rund um die agile Softwareentwicklung.

[Palmer & Felsing 02] Stephen R. Palmer, John M. Felsing: A Practical Guide to the Feature-Driven Development. Prentice Hall, 2002.
Einführung in Feature Driven Development (FDD).

[Schwaber 04] Ken Schwaber: Agile Project Management with Scrum. Microsoft Press, 2004.
Neuere Beschreibung von Scrum mit Schwerpunkt auf Projektmanagement.

[Scrum] <http://www.controlchaos.com>
Zentrale Website rund um Scrum.

[Westphal 05] Frank Westphal: eXtreme Programming (und andere einführende Texte), <http://www.frankwestphal.de>, 2005.
Frank Westphal veröffentlicht auf seiner Website eine Reihe von einleitenden Texten zu verschiedenen Themen rund um eXtreme Programming. Sowohl die kurze Einleitung zu eXtreme Programming selbst als auch »Der Weg zu XP« und einführende Worte zum Umgang mit Unit Testing sind für den Einsteiger empfehlenswert. Zudem finden sich dort noch eine Reihe von nützlichen Links und Buch-Tipps. Seit 2005 veröffentlicht Frank Westphal auf seiner Website den Tonabnehmer, in dem Interviews zu agilen Themen als MP3-Dateien zum Download angeboten werden.

[XP-Mailingliste] xp-forum@yahoogroups.com
Deutschsprache Mailingliste zu eXtreme Programming und anderen agilen Methoden.

[XP-UG-HH] xpug-hh@yahoogroups.com

Mailingliste der XP-User-Group Hamburg.

[XPBuch] <http://community.it-agile.de>

Interaktives Web-System zu diesem Buch. Hier finden Diskussionen rund um agile Softwareentwicklung im Allgemeinen und dieses Buch im Speziellen statt. Die Autoren dieses Buches sind regelmäßige Besucher der Website und geben gerne ihre Erfahrungen weiter.

[XProgramming] <http://www.xprogramming.com>

Website rund um eXtreme Programming (XP).