Jutta Eckstein

XP – eXtreme Programming

Ein leichtgewichtiger Software-Entwicklungsprozess

Die Vorgehensweise bei der Software-Entwicklung wird immer wieder diskutiert. Nachdem sich das Wasserfall-Modell nicht bewährt hat, gibt es nun auch Prozesse, die besser geeignet sind, auf die Änderungswünsche der Kunden zu reagieren. Dieser Artikel stellt eine extreme Vorgehensweise vor und erläutert ihre Grundprinzipien.

Überblick

Thema

Mit eXtreme Programming (XP) haben Kent Beck, Ron Jeffries und Ward Cunnigham einen leichtgewichtigen Software-Entwicklungsprozess vorgestellt. Er kombiniert altbewährte Praktiken in einer Form, die Flexibilität und Qualität von Anwendungen in den Vordergrund stellt.

Statt auf schwere „Tool-Geschütze“ setzt XP auf vergleichsweise einfache Techniken und Grundsätze und betont die Rolle des Programmierers.

XP steht für eine Vision, in der die Einhaltung von Deadlines und 40-Stunden-Wochen kein Widerspruch sind.

Technik

Benutzer- und Test-getriebene Entwicklung unter Verwendung von: Pair Programming, Collective Code Ownership, Continuous Integration und Simple Design.

Voraussetzungen

Kleine bis mittlere Teams; Teammitarbeiter sind räumlich nicht getrennt; ständiges Testen ist möglich

Zielgruppe

Software-Entwickler und Projektleiter

Schwergewichtige Prozesse wie der „Wasserfall” bieten nicht die Flexibilität, die heutzutage von Softwareentwicklung erwartet wird. Auch wenn die Anforderungen an ein System vorab genau spezifiziert werden, so zeigt die Erfahrung, dass sich diese Anforderungen im Laufe der Applikationsentwicklung ändern. Dies wird oft noch dadurch forciert, dass der Anwender erst dann eine Vorstellung von den Möglichkeiten und der Machbarkeit einer Anwendung erhält, wenn er das System zum ersten Mal in der Realität gesehen hat.

Unified Process

Aus diesem Grund wird der Ruf nach leichtgewichtigen Prozessen, die dieser Anforderung Rechnung tragen, immer lauter. Eine Entwicklung in diesem Bereich ist der Unified Process, entwickelt von den „drei Amigos“ – Grady Booch, Ivar Jacobsen und James Rumbaugh. Der Unified Process ist ein Framework, das für jedes Projekt hinsichtlich Teamgröße, Kultur und Projektart konfiguriert werden muss. Trotzdem gehört dieser Prozess zu den schwergewichtigen Prozessen, weil er den Entwicklern und Kunden nicht allzu viel Freiheit lässt.

eXtreme Programming

Dem Unified Process steht das Konzept des eXtreme Programming (XP) gegenüber, das sowohl den Kunden und seine Änderungswünsche als auch die Möglichkeiten der Entwickler berücksichtigt. Vor allem der hohe Stellenwert der Änderungswünsche macht XP zu einem leichtgewichtigen und flexiblen Prozess.

Natürlich müssen auch bei XP einige Regeln eingehalten werden, damit die Applikation am Ende die gewünschte Qualität aufweist und alle Kundenwünsche umsetzt. Deshalb folgt XP bestimmten Grundprinzipien.

„Develop for today“

Zwei Forderungen motivierten die Entwicklung von XP:

1. *Develop for today* – Man konzentriert sich auf die Probleme, die heute anstehen, ohne Probleme vorwegzunehmen, die später auftreten könnten. Das hätte ja nur dann Nutzen, wenn wir bereits heute wüssten, welche Änderungen in Zukunft eintreffen werden. Sollten diese Änderungen später nicht eintreffen, hätten wir weiterhin den architektonischen Ballast in unserem Design. Entwicklung auf XP-Art bedeutet deshalb, sich ausschließlich auf die aktuellen Probleme zu konzentrieren. Es wird davon ausgegangen, dass eine Änderung jederzeit und mit vertretbarem Aufwand eingearbeitet werden kann.

Das ist jedoch nur dann möglich, wenn die zweite Forderung berücksichtigt wird:

2. *Do the simplest thing that could possibly work.* Das bedeutet, dass immer das einfachste Design *(Simple Design)* verwendet wird, um ein Problem zu lösen. *Simple Design* enthält verschiedene Aspekte:

- *Erfüllung der Anforderungen*. Das Design muss der Aufgabe gerecht werden, wie sie zu diesem bestimmten Zeitpunkt verstanden wird.
- *Redundanzfreiheit*. Jede Information darf nur einmal gehalten werden.
- *Refactoring*. Das Design besteht aus der geringst möglichen Anzahl an Klassen, Attributen und Methoden. Das erfordert natürlich, dass sich das Design ständig im Fluss befindet. Man spricht hier auch vom *kontinuierlichen Refactoring* (s. [1] und den Bericht über Patterns in diesem Heft): Wann immer das Design zu komplex wird, sollte es unter Beachtung der oben angeführten Aspekte vereinfacht werden.

Was kostet eine Änderung?

Nur ein einfaches Design erlaubt, Anforderungen, die später auftreten, auch erst zu diesem späteren Zeitpunkt zu berücksichtigen, ohne dass die Kosten dabei linear oder gar exponenziell ansteigen.

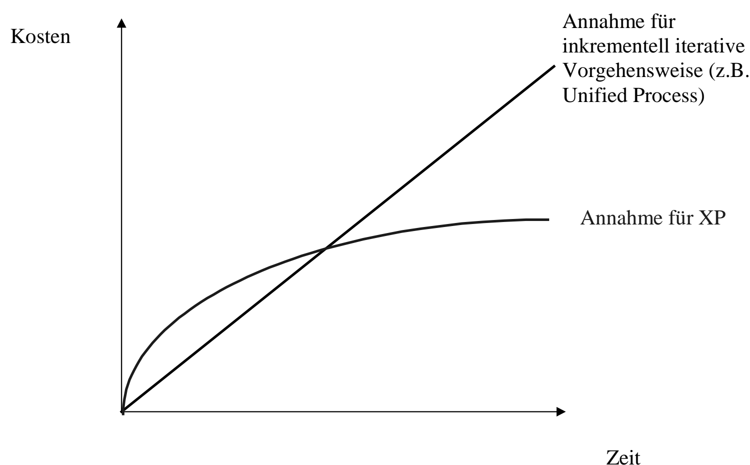

Abbildung 1: Kosten für Änderungen bei unterschiedlichen Vorgehensweisen. Nach der landläufigen Meinung ist eine Änderung in einem Projekt umso teurer, je später sie vorgenommen wird. XP geht davon aus, dass diese Annahme überholt ist.

Oftmals wird sogar von einem exponenziellen Verlauf der Kostenkurve beim Wasserfall-Modell ausgegangen. Ausgangspunkt für diese Annahme sind immer folgende Überlegungen: Wie viel kostet eine Änderung, die während der Analysephase auftritt, wie viel kostet eine Änderung während des Designs und wie viel während der Implementierung? Mit diesen Werten werden die Kurven aufgetragen. Es handelt sich jedoch immer um Schätzungen.

XP geht davon aus, dass die Werkzeuge, insbesondere die der Entwicklungsumgebungen, inzwischen so weit entwickelt sind, dass Änderungen, die relativ spät im Entwicklungszyklus auftreten, keine wesentlich höheren Kosten verursachen als Änderungen in einer früheren Phase *(Abbildung 1)*. Eine weitere Voraussetzung für die geringen Kosten ist natürlich das einfache Design.

Kurze Entwicklungs-/Iterationszyklen

XP geht von sehr kurzen Entwicklungszyklen auf verschiedenen Ebenen aus, wobei grundsätzlich zwischen einem Release, einer Iteration und einer Task unterschieden wird. Die Zeiteinteilung für diese Meilensteine ist wie folgt:

- *Release:* Ein Release ist eine (aus Anwendersicht) lauffähige Version, die eine bestimmte, vorher genau definierte Funktionalität beinhaltet. Für das erste Release werden 3-6 Monate berechnet. Alle weiteren Releases umfassen eine Zeitspanne von 1-3 Monaten.

- *Iteration*: Ein Release besteht aus mehreren Iterationen, die sich über 1-4 Wochen erstrecken.
- *Task*: Eine Iteration wiederum umfasst mehrere Tasks, wobei eine Task in 1-3 Tagen bearbeitet wird.

Abbildung 2 zeigt, wie sich die Entwicklungszyklen bei den verschiedenen Vorgehensweisen unterscheiden.

Sie sehen auch, dass bei XP die Phasen gekippt sind, d.h. Analyse, Design, Implementierung und Test werden nicht als voneinander abgegrenzte Phasen aufgefasst, sondern als parallel ablaufende Aktivitäten.

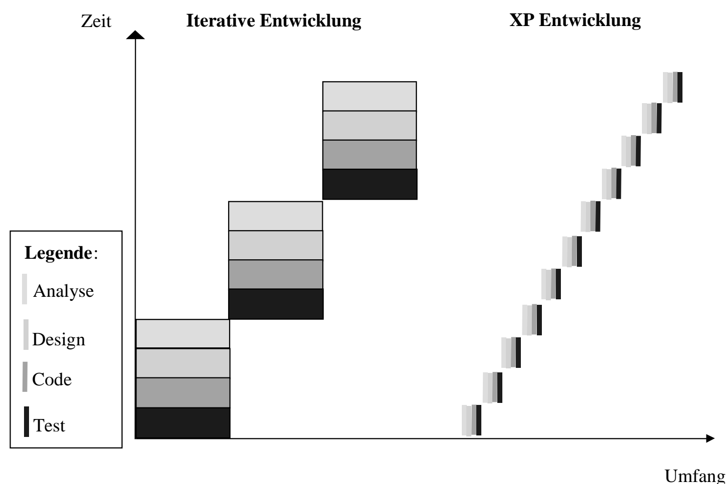

Abbildung 2: Entwicklungszyklen nach dem XP-Ansatz im Vergleich zu einer herkömmlichen Vorgehensweise. Analyse, Design, Implementierung und Test laufen parallel und in kleineren Schritten ab. Die Möglichkeit, auf geänderte Anforderungen schnell zu reagieren, ist damit sozusagen in XP „eingebaut“.

XP-Werkzeuge & -Prinzipien

Basierend auf diesen Grundsätzen wurden die Praktiken und Prinzipien entwickelt, die den XP-Prozess bestimmen. Alle Prinzipien sind (mehr oder weniger stark) voneinander abhängig und beeinflussen sich gegenseitig.

User Storys

Die Anforderungen schreibt im Idealfall der Kunde selbst in so genannten *User Storys* fest. Die User Story beschreibt von außen, was der Kunde von einem System erwartet bzw. was er mit dem System tun möchte (ähnlich wie bei den UML Use Cases). Diese Storys sind die zentrale Einheit, die das gesamte weitere Vorgehen steuern.

Gewöhnlich notiert man je eine User Story auf eine Karteikarte. So ergeben sich meist ganz natürlich Funktionalitätseinheiten, die innerhalb einer Task realisiert werden können.

The Planning Game

Basierend auf den User Storys werden die Releases geplant. Dies geschieht mit Hilfe des so genannten *Planning Game*. Dabei sortiert der Kunde (!) in der ersten Runde die User Storys in drei Stapel, je nach Priorität:

1. In den ersten Stapel kommen User Storys, die essenziell sind: werden diese Funktionen nicht implementiert, so ist das gesamte Release hinfällig.
2. Der zweite Stapel enthält User Storys, die nicht essenziell sind, die jedoch den Wert des Release signifikant erhöhen.
3. Im dritten Stapel liegen die User Storys, bei denen es prima wäre, wenn sie erfüllt wären *(nice to have)*, die aber geringe Priorität haben.

In der zweiten Runde des Planning Game treten die Entwickler in Aktion. Sie geben an, wie viel Zeit benötigt wird, um die einzelnen User Storys zu implementieren. Dabei rechnen sie in Idealtagen, d.h. es wird davon ausgegangen, dass die Entwickler hundert Prozent ihrer Zeit für die Implementierung verwenden können. Kann eine User Story nicht geschätzt werden, dann geht sie zurück an den Kunden. Er soll diese User Story genauer spezifizieren, eventuell mit der Hilfe eines Entwicklers.

Wenn alle User Storys geschätzt wurden, wird das nächste Release mit dem Kunden verhandelt. Da für ein Release 1-3 Monate Zeit zur Verfügung steht (außer beim ersten Release), kann man genau angeben, wie viele User Storys in dieser Zeit umgesetzt werden können. Es liegt dann am Kunden zu sagen, welche User Storys er in diesem Release implementiert haben möchte. Das heißt, das Planning Game ist eine Art Verhandlung über die Inhalte des nächsten Release. (Natürlich entwickelt kein Entwickler in Idealtagen. Vor der Verhandlung multipliziert man daher die geschätzten Idealtage mit einem teamspezifischen Faktor *(Load Factor)*, der typischerweise im Bereich 1,5 bis 3 liegt. Aus 5 geschätzten Idealtagen werden damit schnell 2 bis 3 reale 40-Stunden-Arbeitswochen.)

Es ist wichtig zu betonen, dass auf der einen Seite der Kunde immer Einfluss darauf hat, was in das nächste Release aufgenommen werden soll, und auf der anderen Seite die Entwickler eine Schätzung abgeben, wie lange sie für ein Feature benötigen. Damit übernehmen die Entwickler die Verantwortung, ihrer eigenen Schätzung gerecht zu werden. Oft ist es ja üblich, dass sie z.B. vom Management eine Zeitvorgabe erhalten, die nicht mit ihren Möglichkeiten in Einklang steht (vgl. *Kasten 1*).

Das Planning Game begleitet den gesamten Entwicklungsprozess, da jederzeit neue User Storys auftauchen können, die im nächsten Release berücksichtigt werden sollen. Oft werden sie aber auch direkt im aktuellen Release berücksichtigt; dafür fällt dann natürlich eine andere User Story mit gleichem Umfang heraus.

Man kann nicht alles vorschreiben

Zu den Verdiensten von XP gehört es, Auftraggebern bzw. dem Management klar gemacht zu haben, dass sie nicht alle Projektparameter bestimmen können. Im Internet finden Sie dazu unter [2] eine ausführliche und einleuchtende Geschichte; kurz und knapp lässt sich die XP-Erkenntnis jedoch auch so zusammenfassen: *You can't have the cake and eat it*. Das Management kann einfach nicht alles gleichzeitig haben.

Von den vier Projektparametern Kosten/Budget, Zeit, Qualität und Funktionalität/Features (Scope) können immer nur drei vom Management bestimmt werden, den letzten Parameter definieren Sie als Entwickler. Zwei Beispiele:

1. Das Management bestimmt das Budget, macht eine Zeitvorgabe und setzt eine bestimmte Funktionalität fest. Dann ist es an Ihnen, die Qualität zu definieren, die Sie unter den gegebenen Parameterwerten liefern können. Bei kleinem Budget, wenig Zeit und vielen Funktionen kann die Qualität einfach nicht hoch sein.
2. Das Management macht eine Zeitvorgabe, definiert die Qualität und setzt einen Kostenrahmen. Dann bestimmen Sie, wie viel und welche Funktionalität Sie unter diesen Umständen realisieren können.

Fühlen Sie sich also in Zukunft nicht einfach unwohl in einem Projekt, sondern legen Sie den Finger in die Wunde: Wahrscheinlich hat Ihr Management wieder zu viel bestimmen wollen. Geben Sie den Chefs dann am besten [2] zu lesen!

Seien Sie jedoch vorsichtig: Vielleicht hat man Ihnen bisher Verantwortung für etwas zugewiesen, das Sie nicht selbst bestimmen durften. Zu Recht haben Sie dann diese Verantwortung (innerlich) abgelehnt oder haben sich zumindest schlecht damit gefühlt. Verantwortung kann nicht einfach zugewiesen oder befohlen werden. Verantwortung kann man nur selbst und bewusst übernehmen.

Sobald Ihr Management sich jedoch darauf einlässt, nur noch drei der vier Projektparameter zu bestimmen, übernehmen Sie tatsächlich Verantwortung für den vierten Parameter, den Sie selbst festlegen. Schätzen Sie also sehr überlegt und eher konservativ – und bestimmen Sie den *Load Factor* für sich bzw. Ihr Team.

Kasten 1: Vier Parameter bestimmen den Fortgang eines Softwareprojekts.

System-Metapher

Unter der System-Metapher wird die Grundarchitektur des Systems verstanden, oder anders ausgedrückt: das Gesamtbild, das alle Entwickler vor Augen haben. Alle Entwickler kennen nicht nur den Teilbereich oder das Subsystem, an dem sie selbst arbeiten, sondern auch das Gesamtziel und die Grundarchitektur, die das System bestimmen. Im Unified Process wird hier von der Architektur-Referenzlinie gesprochen *(architecture baseline).*

Coding Standards

Während der Kunde die User Storys schreibt, definieren die Entwickler die Kodier-Richtlinien. Diese Richtlinien sind die Basis für alles Weitere. Nur wenn alle diese Richtlinien kennen und einhalten, können die weiteren Aktivitäten erfolgreich sein. Im Gegensatz zu anderen Vorgehensweisen definieren jedoch die Entwickler diese Richtlinien selbst und bekommen sie nicht von außen aufgezwungen. Nur dadurch besteht die Chance, dass sich auch wirklich alle an diese Richtlinien halten.

Iteration und Task Planning

Wie bereits erläutert wurde, besteht ein Release aus mehreren Iterationen und eine Iteration wiederum aus mehreren Tasks *(Abbildung 3)*. Bei den Planungsaktivitäten werden nun die User Storys des aktuellen Release in kleinere Abschnitte aufgeteilt – zuerst in Iterationen und diese wiederum in Tasks.

Nun nimmt sich jeder Entwickler eine Task vor und schätzt zunächst, wie lange er brauchen wird, um diese Task umzusetzen. So sieht man schon relativ früh, ob die Schätzung für das gesamte Release noch gültig ist; zum anderen wird sichergestellt, dass einem Entwickler

nicht zu viel bzw. zu wenig zugemutet wird. Da diese Schätzung vom Entwickler selbst kommt, kann man sicher sein, dass er alles unternimmt, um diese Zeiten einzuhalten.

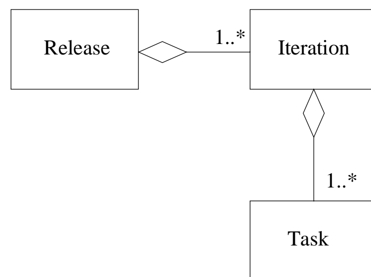

Abbildung 3: Der Zusammenhang von Releases, Iteration und Tasks. Tasks sind die kleinste Einheit, deren Aufwand konkret von einem Entwickler geschätzt wird. Der Aufwand für eine Iteration bzw. ein Release ergibt sich aus der Summe von Tasks. So bestimmen letztlich die Entwickler, welche Tasks bzw. User Storys in einem Release enthalten sein werden/können.

Pair Programming

Zur Umsetzung einer Task werden immer zwei (!) Entwickler benötigt. Diese Vorgehensweise hat verschiedene Effekte:

- Es wird sichergestellt, dass die Kodierrichtlinien eingehalten werden, da der nicht kodierende Entwickler direkt einen Code Review macht.
- Die Qualität wird überwacht, der Code Review findet sozusagen online statt, da sich der nicht programmierende Partner auf einer anderen „Ebene“ befindet und größere Zusammenhänge überschauen oder Fehler besser erkennen kann.
- Außerdem kennen immer mindestens zwei Entwickler den Code, d.h. sollte ein Entwickler bei Fragen nicht zugegen sein oder sogar das Team verlassen haben, so gibt es immer noch einen zweiten, der sich damit auskennt.

Weiterhin kann so ein kleines Team eher sicherstellen, dass tatsächlich das *„simplest thing that could possibly work“* implementiert wird und nicht irgendwelche abenteuerlichen Designs.

Die Paar- oder Partnerprogrammierung ist allerdings eines der umstrittensten Merkmale von XP. Das Management sieht darin meist nur eine Verschwendung von Ressourcen. Leider gibt es noch keine empirischen Studien darüber, welchen Zusammenhang Partnerprogrammierung und Produktivität/Qualität tatsächlich haben. Die Berichte von großen Vorteilen sind eher anekdotisch, wenn auch durchaus glaubhaft. Auch gibt es erste Anzeichen in der XP-Gemeinde, den „Zwang“ zur Partnerprogrammierung zu lockern. Während des Prototypings z.B. kann auch ein Entwickler allein arbeiten.

Unabhängig davon sollten Sie Partnerprogrammierung durchaus einmal ausprobieren. Vielleicht verlieren Sie dabei Zeit, doch sicher gewinnen Sie Teamgeist und Qualität.

Unit Tests

Auch die Vorgehensweise bei der Implementierung ist klar definiert. So werden die Tests für eine Task grundsätzlich vor (!) der Entwicklung dieser Task geschrieben (vgl. [3]). So wird man sich vorher darüber klar, wie Schnittstellen aussehen sollen und was die Task eigentlich leisten soll.

Gleichzeitig hat man nachher eine Kontrolle, ob die Implementierung auch der Aufgabe gerecht wird. Auch bei späteren Änderungen ist man damit auf der sicheren Seite, da man immer testen kann, ob die Grundfunktionalität noch erfüllt wird. Zur Unterstützung dieser Unit Tests stehen Frameworks zum freien Herunterladen für unterschiedliche Programmiersprachen zur Verfügung (s.a. *VBUnit* auf der Heft-CD und den Code zu [3]).

Functional Tests

Während die Entwickler das aktuelle Release in Form einzelner Tasks implementieren, entwickelt der Kunde die Funktionstests. Wie das geschieht, hängt vom Kenntnisstand des Kunden und der bereitgestellten Entwicklungsumgebung ab. Entweder entwickelt der Kunde die Tests direkt selbst, oder mit Hilfe einer Skriptsprache, die diese Implementierung vereinfacht, oder ihm wird ein Entwickler zur Seite gestellt, der die Funktionstests mit dem Kunden zusammen umsetzt.

Die Funktionstests sollen natürlich sicherstellen, dass das Release auch den Anforderungen gerecht wird. Außerdem sorgen sie bei späteren Releases dafür, dass die bereits bestehende Funktionalität nicht von der Weiterentwicklung negativ beeinflusst wird. Spätestens jetzt sollte klar sein, welche große Rolle Tests bei XP spielen.

Collective Code Ownership und Refactoring

Bei der Umsetzung einer Task bleibt es nicht aus, dass man Code verwendet, der von einem anderen Entwicklerpaar stammt. Stellt man dabei fest, dass dieser Code z.B. nicht dem Grundsatz des einfachen Designs entspricht oder Ungereimtheiten enthält, so sollen diese Probleme sofort behoben werden (durch *Refactoring*, s. [1]). Das heißt, der Code gehört nicht dem Entwicklerpaar, das ihn erstellt hat, sondern immer dem gesamten Team. Die bestehenden und neuen Tests stellen sicher, dass der Code immer noch seine Grundfunktionalität beibehält. Auch die paarweise Entwicklung sorgt dafür, dass nicht ein Entwickler einfach wild drauflos ändert.

Wenn es sehr aufwendig ist, das vorliegende Design zu bereinigen, so wird dies auf einer Karteikarte vermerkt. Damit erhält man neben den Tasks, die aus den Anforderungen des Kunden entstehen, auch so genannte *Long-Term-Refactoring-Goals*, die wie Tasks behandelt werden. Auch diese Tasks werden in Releases eingeplant.

Continuous Integration

Das Ziel ist, ein ständig (!) lauffähiges System zu haben. Aus diesem Grund werden alle Tasks, die die Tests erfolgreich durchlaufen haben, in das Gesamtsystem integriert. Integration steht also nicht am Ende der Gesamtentwicklung, sondern gehört mit in den normalen Entwicklungszyklus. Meist gibt es dafür eine eigene Integrationsmaschine. Sobald man seinen eigenen Code integriert hat, müssen zur Qualitätssicherung alle Tests durchlaufen werden. Falls dabei ein Problem entdeckt wird, ist eindeutig die Integration der letzten Task der Auslöser – die Fehlersuche wird dadurch eingeschränkt. An der Integrationsmaschine führt der Kunde dann auch seine Funktionstests durch.

Metrics

Der gesamte Zyklus zur Entwicklung eines Release wird ständig dahingehend überwacht, ob die Schätzungen auch ihre Richtigkeit haben. Gegebenenfalls muss man neu darüber verhandeln, wie viele User Storys in diesem Release umgesetzt werden.

Vor allem aber dient die Überwachung dazu, bei den nächsten Schätzungen der Realität näher zu kommen. Aus dem Unterschied zwischen geschätzter Zeit und tatsächlicher Dauer ergibt sich der Korrekturfaktor *(Load Factor)*, mit dem man zukünftig die Schätzungen der Entwickler korrigieren kann.

Voraussetzungen und Grenzbereiche

Voraussetzungen

Da *Kommunikation* ein zentraler Bestandteil von XP ist, muss auch die Arbeitsumgebung entsprechend eingerichtet sein. Es ist unabdingbar, dass die Teammitarbeiter im gleichen Raum arbeiten. Manchmal heißt es sogar, das wichtigste Werkzeug in einem XP-Projekt sei der Schraubenzieher: zur Umgestaltung der Räume.

Im Mittelpunkt einer mit XP entwickelten Applikation steht immer ihre Qualität. So muss die gesamte Anwendung immer allen Tests gerecht werden und dabei gleichzeitig eine Architektur aufweisen, die einfach und nachvollziehbar ist und keine Redundanzen enthält. Vor allem Letzteres wird im Visual Basic-Umfeld immer wieder diskutiert: Eine Form, Redundan-

zen zu vermeiden, ist die Vererbung der Implementierung. Sprachen wie Java oder Smalltalk unterstützen das, Visual Basic jedoch nicht.

Für den XP-Prozess muss die *Software-Entwicklungsumgebung* das inkrementelle Arbeiten ermöglichen. Das ist schwierig, wenn die Entwicklungsumgebung lange Kompilier- und Linkzeiten erfordert. Auch ein vernünftiges Konfigurations- und Versionsmanagement ist wichtig, um die inkrementellen Änderungen zu unterstützen. VB unterstützt eine inkrementelle Entwicklung sehr gut durch die Möglichkeit, COM-Komponenten zu erzeugen.

Die *Hardware* muss ständiges Testen und Integrieren ermöglichen. Manchmal ist es jedoch nicht möglich, den Entwicklern eine genaue Entsprechung der Zielumgebung zur Verfügung zu stellen, z.B. weil diese sehr teuer ist. In diesem Fall kann weder ständig integriert noch getestet werden, d.h. in einer solchen Umgebung ist XP nicht möglich.

Bisher gibt es nur Erfahrung mit *kleinen Teams*. Darum gibt es keine Aussagen darüber, ob XP auch in Teams mit mehr als 15 Mitarbeitern erfolgreich eingesetzt werden kann.

Der Kunde gehört zum Team! XP fordert, dass ein zukünftiger Benutzer oder Fachexperte ständig im Team mitarbeitet. Nur dadurch ist die *Trennung von technischen und geschäftsbedingten Entscheidungen* überhaupt möglich. Dies kommt v.a. beim Planning Game zum Tragen: Was im nächsten Release umgesetzt werden soll, ist eine geschäftsbedingte Entscheidung, die die Entwickler nicht treffen können. Die Schätzungen für die User Storys sind dagegen eine technische Entscheidung, die der Kunde nicht beeinflussen kann. Der Kunde, der immer vor Ort ist, gibt auch immer *direktes Feedback* zum aktuellen Stand der Entwicklung.

Die letzte Voraussetzung mutet vielleicht etwas seltsam an: Überstunden dürfen kein Status sein. XP geht von einer *40-Stunden-Woche* aus, weil Überstunden erfahrungsgemäß zu Leistungsminderung führen. Man kann über einen kürzeren Zeitraum (max. eine Woche) mit Überstunden durchaus produktiver sein, langfristig ist das jedoch nicht möglich. Falls Überstunden im Projekt zur Gewohnheit werden, so liegt das Problem vermutlich an einer anderen Stelle, z.B. daran, dass die Metriken außer Acht gelassen wurden (vgl. auch *Kasten 1*).

Grenzbereiche

In einigen Bereichen gibt es noch nicht genug Erfahrung mit XP, sodass man nicht mit Sicherheit sagen kann, ob es bei solchen Projekten funktioniert:

- **Festpreisprojekte**. Das Angebot richtet sich nach den User Storys. Es sollte kein Problem sein, neue Anforderungen aufzunehmen: Für jede neue Anforderung wird eine andere Anforderung gleichen Umfangs herausgenommen.
- **Framework-Entwicklung**. Bei der Framework-Entwicklung ist kein Kunde mit im Team. Jedoch könnte ein Teammitarbeiter, der bereits eine Applikation in der Zieldomäne des Frameworks entwickelt hat, die Rolle des Kunden übernehmen. Er bestimmt dann, welche Anforderungen im nächsten Release berücksichtigt werden, kümmert sich um die funktionalen Tests, spezifiziert die meisten Anforderungen und legt deren Prioritäten fest. Dabei besteht die größte Gefahr darin, dass technische und geschäftsbedingte Entscheidungen vermischt werden.

Projekte

XP ist noch relativ jung. Das erste Projekt wurde 1997 vorgestellt, damals allerdings noch nicht unter dem Namen XP. Inzwischen wurde XP in mehren Projekten eingesetzt. *Tabelle 1* gibt einen Überblick über drei dieser Projekte, der die Einsatzmöglichkeiten von XP verdeutlichen soll.

Unternehmen	Projekt	Teamgröße	Dauer (in Jahren)	Entwicklungssprache
Acxiom	Campaign Management	10	3	Forté
DaimlerChrysler	Personalabrech-	10	4	Smalltalk

	nung			
StorageTek	Roboter-Steue-rung	10	noch in der Ent-wicklung	Java

Tabelle 1: Übersicht über einige XP-Projekte. Details siehe [5]

Zufälligerweise sind bei den vorgestellten Projekten die Teamgrößen und die Projektdauer nahezu identisch, obwohl sich die Projektart und die Programmiersprachen gravierend unterscheiden.

Weitere XP-Projekte wurden und werden bei folgenden Firmen durchgeführt: Ford Motor, WyCash, First Union National Bank, CSLife, Bayerische Landesbank.

Ein VB-Projekt ist ebenfalls in Entwicklung, allerdings gibt es darüber noch keinen Projektbericht. Die Entwickler wollen ihre Erfahrungen mit XP auf der XP-Homepage [5] veröffentlichen.

Fazit

Mit XP wurde ein leichtgewichtiger Prozess vorgestellt, bei dem die Entwicklung einer Applikation sowie die Kundenwünsche im Vordergrund stehen. XP legt großen Wert darauf, dass Änderungen Bestandteil des Prozesses sind – es gilt als normal, wenn während der Entwicklung Änderungen auftreten. Auch wenn andere Prozesse ebenfalls versuchen, mit Änderungen umzugehen, so werden sie dort doch immer als Ausnahmen betrachtet. Dadurch, dass man Änderungen bei XP gern annimmt *(„Embrace Change“)*, rückt der Kunde oder Benutzer wieder in den Mittelpunkt der Applikationsentwicklung.

Die einzelnen Prinzipien von XP sind nicht wirklich neu. XP stellt altbewährte Praktiken in einen neuen Zusammenhang, um so einen vor allem pragmatischen Prozess zu modellieren.

Quellen

[1] Martin Fowler: *Refactoring – Wie Sie das Design vorhandener Software verbessern;* Addison-Wesley 2000

[2] Once upon a time…: *http://www.xprogramming.com/xpmag/kings_dinner.htm*

[3] Ralf Westphal: *Vorsicht vor Spaß am Testen!;* BasicPro 2/00, S. 23

[4] *http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap*: Homepage von eXtreme Programming.

[5] *http://c2.com/cgi/wiki?ExtremeProgrammingProjects*: Projektberichte.

[6] Kent Beck: *eXtreme Programming eXplained: Embrace Change;* Addison-Wesley 1999

[7] Alistair Cockburn: *Surviving Object-Oriented Projects. A Manager’s Guide;* Addison-Wesley 1998

[8] Andrew Hunt, Dave Thomas: *The Pragmatic Programmer. From Journeyman to Master.* Addison-Wesley 1999

[9] Ivar Jacobsen, Grady Booch, James Rumbaugh: *The Unified Software Development Process*. Addison-Wesley 1998

[10] Hans Wegener: *Extreme Ansichten – Für und Wider des Extreme Programming;* iX 12/99, S. 126

[11] Web-Adressen:

- *www.xprogramming.com*
- *www.extremeprogramming.org*
- *www.ootips.org/xp.html*
- *www.xpdeveloper.com*
- *www.sern.enel.ucalgary.ca/yip/summaries/extreme.html*

Jutta Eckstein ist als unabhängige Beraterin und Trainerin für objektorientierte Technologien tätig. Ihr Interesse an XP basiert auf ihrer zehnjährigen Erfahrung in der Erstellung von objektorientierter Software. Sie ist Mitglied des Programmkomitees der „XP2000 – eXtreme Programming and Flexible Processes in Software Engineering“ in Cagliari, Italien (http://numa.sern.enel.ucalgary.ca/extreme/). Für Fragen und Anregungen erreichen Sie Jutta Eckstein per EMail an jeckstein@acm.org.