

eXtreme Proramming Part II

Steve Moser, Marcus Stegner, Robert Cajer

8. Dezember 2002

Inhaltsverzeichnis

1 eXtreme Programming Teil II	3
1.1 Kurzer Überblick	3
2 Was ist XP?	3
2.1 Warum heist es eXtreme Programming	3
2.2 Was verspricht eXtreme Programming	4
2.3 Was sind die Kennzeichen von XP	4
3 Der Umgang mit Risiken –Das Grundproblem der Softwareentwicklung–	5
3.1 Risiko:Termin nicht einhaltbar	5
3.2 Risiko: Projektabbruch	5
3.3 Risiko: Unwartbarkeit	6
3.4 Risiko: Unbenutzbarkeit durch Programmierfehler	6
3.5 Risiko: Unbenutzbarkeit durch Fehlentwicklung	6
3.6 Risiko: Änderung der Anforderung	7
3.7 Risiko: Featuritis	7
4 Zeit/Kosten Faktor in XP	7
4.1 Wie teuer sind Änderungen?	7
4.2 Wie teuer sind Änderungen?	8
5 Grundwerte XP im Beispiel	8
5.1 Der Anfang allen Seins, die Problemschilderung	8
5.2 Die Umsetzung	8
5.3 Die Codierung	9
5.4 Pair Programming	9
5.5 Schrittweise Annäherung	10
5.6 Grundwerte	11
6 Die Rollenverteilung -Die Spieler in XP-	12
6.1 Der Truckfaktor	12
6.2 Rollen	12
6.2.1 Auftraggeber (Client)	12
6.2.2 Kunde (Customer)	13
6.2.3 Programmierer (Programmer)	13
6.2.4 Tester	13
6.2.5 Verfolger (Tracker)	14

6.2.6	Trainer (Coach)	14
7	Der Weg zu einer Lösung	14
7.1	Einflußfaktoren	14
7.2	Einfluß des Kostenrahmens	14
7.3	Einfluss des Zeitrahmens	15
7.4	Einfluss der Qualität	15
7.5	Einfluss des Funktions-Umfangs	16
7.6	Einflussfaktoren: Fazit/Empfehlungen	16
8	Wann ist XP nicht sinnvoll?	16
9	Quellennachweis	17
10	Anhang	17

1 eXtreme Programming Teil II

1.1 Kurzer Überblick

Der Informatik wird oft vorgeworfen, sie habe noch kein Mittel gefunden, Software 'in Time' und 'in Budget' zu liefern. Wissenschaftliche Untersuchungen untermauern diesen Vorwurf, (eine erstmals 1995 veröffentlichte Studie der Standish Group). Dieser jährlich erneuerte Bericht fördert zu Tage, dass Softwareprojekte Kosten, Zeit oder sogar beides in erheblichem Maße überschreitet.

Aus dieser Erkenntnis suchten Entwickler und Forscher nach Gründen für dieses offensichtliche Versagen. Sie fanden heraus, dass nicht technische Mängel, sondern mangelnde Kundeneinbindung, methodische Defizite, sowie Unzulänglichkeiten im Projektmanagement die meisten Entwicklungsvorhaben zum Scheitern bringen. Hier soll nun eine Methode dargestellt werden, mit denen sich solche Probleme vermeiden lassen - sollten - diese Methode nennt sich Extreme Programming.

Mit eXtreme Programming haben Kent Beck, Ron Jeffries und Ward Cunningham 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 und des Kunden. XP steht für eine Vision in der die Einhaltung von Deadlines und 40 Stunden Wochen kein Widerspruch sind.

Technik:

Benutzer und Testgeschriebene Entwicklung unter Verwendung von:

- Pair Programming
- Collective Code Ownership
- Continuous Integration
- Simple Design

Voraussetzungen:

Kleine bis mittlere Teams, Teammitarbeiter sind räumlich nicht voneinander getrennt. Ständiges Testen ist möglich.

2 Was ist XP?

2.1 Warum heist es eXtreme Programming

Es gibt hier einen Leitsatz der in der Literatur seinen Einzug gefunden hat:

"Extreme Programming takes common principles and practices to extreme levels."

Unter "common principles and practices" versteht man in erster Linie, dass der Projektablauf auf üblicher Basis sequentiell gehalten wird. Man beginnt mit der Analyse der Anforderungen, schreitet zum Entwurf und schließlich zum Implementieren. Danach wird getestet und letztendlich geht das Projekt in Betrieb.

Bei XP versucht man diese Phasen parallel zu halten.

Man weiss aus Erfahrung dass Code Reviews gut sind, also überarbeitet man den Code in XP über das Pair Programming fortlaufend.

Wenn man weiss, dass Testen wichtig ist, dann testet man in XP über den ganzen Projektablauf hinaus. Zum einen über den Programmierer, der seine Modultests ausführt, und zum anderen durch den Kunden, der die Funktionstests durchführt.

Wenn man "Code Reviews" betreibt und den geschriebenen Code für gut befunden und ausgiebig getestet hat, muss er in das Projekt integriert werden. Dadurch dass der Code mehrmals am Tag getestet und überarbeitet wird, finden diese Integrationstests mehrmals am Tag statt. Dies nennt man dann "Continuous Integration"

Das Design sollte so einfach bzw. so komplex wie nötig sein, d.h. das Design besteht aus der geringst möglichen Anzahl an Klassen, Attributen und Methoden. Das erfordert, dass sich das Design ständig im Fluss befindet. Man spricht hier vom "Refactoring". Wann immer das Design zu komplex wird, sollte es nach den vorher genannten Aspekten vereinfacht werden.

Um im Projekt den Überblick zu behalten, werden die einzelnen Releases in Teilaufgaben (Iterationen) aufgeteilt. Nun könnte man behaupten: "Das ist doch normal!" XP setzt hier aber voraus, dass diese Iterationen sehr kurz gehalten werden. Man plant hier Minuten bis Stunden ein ("planning game"), und nicht wie herkömmlich Wochen oder sogar Monate.

Diese Vorgehensweise in ihrer Gesamtheit ist eXtrem.

2.2 Was verspricht eXtreme Programming

Für den Programmierer verspricht XP dass er sich auf die wirklich relevanten Dinge konzentrieren kann und dass er nicht allein ist in schwierigen Situationen. (Stichwort "Pair Programming")

Kunden und Managern sehen den großen Vorteil darin, dass sie den maximalen Output aus einer Projektwoche mit XP erzielen. Sie bekommen konkrete Ergebnisse in kurzen Intervallen. XP soll hier mit kurzen Reaktionszeiten bei Änderungsanforderungen aufwarten. Und etwas, was wichtig ist für Kunde und Manager: Es soll mit einem vertretbaren Aufwand realisiert werden können.

2.3 Was sind die Kennzeichen von XP

Zum einen zeichnet sich XP durch seine kurzen Zyklen aus, die natürlich ein schnelles Feedback zur Folge haben. Ein weiterer Punkt ist das inkrementelle Planen. Der anfangs einfache Plan wird laufend weiterentwickelt. Hierdurch entsteht ein evolutionärer Entwurf, solange das System im Einsatz ist. Ein anderes wesentliches Kennzeichen ist, dass bei XP fortlaufend getestet wird. Das Testen wird automatisiert, was den großen Vorteil hat, dass eine frühe Fehlererkennung möglich ist. Durch die enge Zusammenarbeit von "normalen" Programmierern, also keine Überflieger oder Spezialisten, findet ein Einklang zwischen den persönlichen, kurzfristigen "Instinkten" der Programmierer statt, was den übergeordneten langfristigen Projektzielen dient.

3 Der Umgang mit Risiken –Das Grundproblem der Softwareentwicklung–

3.1 Risiko: Termin nicht einhaltbar

Wer in einem Projekt jemals mitgearbeitet hat, kam vielleicht schon mal in Bedrängnis dass er am Auslieferungstag dem Kunden mitteilen muss, dass sein Produkt noch 6 Monate braucht um einsatzfähig zu sein.

Nun stellt man sich die Frage ob man dieses Problem mit XP eingrenzen kann oder besser sogar, lösen kann.

Eingrenzen? Definitiv ja. Lösen? Nein.

Um dieses Problem in den Griff zu bekommen, setzt XP auf kurze Zyklen, d.h. "Product Releases" werden innerhalb Wochen, höchstens Monaten abgeschlossen. Die Iterationen innerhalb eines Releases sollen in ein bis vier Wochen realisiert werden. Und für Programmieraufgaben innerhalb einer Iteration (Tasks) werden ein bis drei Tage veranschlagt. Natürlich können diese Zeitintervalle dem jeweiligen Projekt angepasst werden.

Funktionen, die für den Kunden eine hohe Priorität aufweisen, werden als erste implementiert. Evtl. muss der Kunde hier in Kauf nehmen, dass die Gesamtfunktionalität darunter leidet. Doch lieber ein Produkt, das garantiert funktioniert und vielleicht nicht alles kann, als ein Produkt, das weder zum Auslieferungstag fertig wird, noch ein Produkt, das gegen Ende des Projektes hin, nur nach dem Motto "quick and dirty" realisiert wird.

Welchen Vorteil hat diese Vorgehensweise?

Zum einen, dass nur das was gewünscht und als äußerst wichtig eingestuft, implementiert wird. Zum anderen, dadurch dass der Kunde involviert ist (dazu später mehr) erhält der Programmierer ein Feedback und weiss wesentlich schneller ob seine Funktionen das leisten was der Kunde sich erhofft, was auch dazu führt, dass Probleme früher erkannt werden können. Desweiteren tauchen keine unangenehmen Überraschungen von Funktionen auf, die man aufgrund ihrer geringeren Priorität weggelassen hat.

3.2 Risiko: Projektabbruch

Es kann bei einem Projekt zum Abbruch kommen, wenn es z.B. laufend verzögert wird, oder weil die Aufwandsschätzung absolut daneben lag und nun die finanziellen Mittel ausgehen.

XP versucht hier in Zusammenarbeit mit dem Kunden ein für ihn bestes Aufwand/Nutzen-Verhältnis zu ermitteln. Der Kunde gleicht seine Anforderungen mit den Aufwandsabschätzungen der Programmierer ab. Der Kunde weiss somit im voraus, was er für seine Investition erwarten kann, und vor allem erwarten ihn keine Überraschungen, da er am Projektablauf mitintegriert ist. Auch kann vor der Inbetriebnahme der Software weniger schiefgehen.

3.3 Risiko: Unwartbarkeit

Das System ist effektiv im Einsatz, doch aufgrund zu hoher Wartungskosten wird es abgelöst.

Um diesem Risiko entgegen zu wirken, setzt XP auf permanentes Testen in der Entwicklungsphase. Und zwar durch beide Seiten- dem Programmierer und dem Kunden selbst. Der Programmierer ist für seine Modultests verantwortlich, der Kunde übernimmt hier die Rolle (s.u.a. Rollenverteilung) eines Funktionstesters, oder sogar gleichzeitig die des Endverbraucher, wenn dieser nicht miteingebunden wird. Man kann sich das folgendermaßen vorstellen:

Der Kunde gibt vor, dass eine Funktion den Gesamturlaub des Jahres eines Angestellten ausgeben soll. Der Programmierer schreibt seine Funktion und diese wird nach ausführlichen Tests in das Design integriert. Der Kunde testet nun ob die Funktion seinen Erwartungen entspricht. Ist z.B. diese Funktion für ihn so wichtig, dass er sie auf einen Klick bedienen kann, ist sie im Hauptfenster über einen Radio Button sofort erreichbar, oder ist es eher ein nebenläufiges Feature, das zwar mitintegriert werden muss (Stichwort Priorität), aber nicht sofort abrufbar sein muss, dann kann dies über ein Untermenü gesteuert werden? Wie soll die Übersicht gegliedert sein?...

Dies mag zwar ein banales Beispiel sein, doch macht es deutlich wie sehr der Kunde in das Projekt miteingebunden wird.

Durch diesen Ansatz wird im Vorfeld Fehlern, im Sinne von Programmierfehlern, vorgebeugt. Auch ist das System jederzeit in Bestform. Das System muss somit im Nachhinein nicht dauernd gewartet werden.

3.4 Risiko: Unbenutzbarkeit durch Programmierfehler

Das System wird in Betrieb genommen, doch es wird nicht angenommen da es zu viele Programmierfehler enthält.

(s.Punkt 3.3)

3.5 Risiko: Unbenutzbarkeit durch Fehlentwicklung

Was tun, wenn das System beim Kunden etwas anderes tut, als der Kunde wollte?

XP geht den Weg des geringsten Widerstandes. Es integriert den Kunden im Projekt und gibt ihm eine aktive Rolle. Wie erwähnt kümmert er sich um die Anforderungsspezifikationen und übernimmt anschließend die Funktionstests. Das große Plus dieser Sichtweise ist, dass durch andauerndes Feedback, sowohl von den Programmierern zum Kunden, und umgekehrt, Kommunikationsprobleme sehr schnell im Ansatz gelöst werden können. Ein Kunde der den Fortschritt miterlebt und mitgestaltet, ist ein zufriedener Kunde. Auch überarbeitet er automatisch seine Anforderungen, anhand dessen dass er sieht, welcher Aufwand für ein bestimmtes Feature betrieben werden muss und setzt so neue Prioritäten. Wenn das System im Betrieb ist, kann man zu 90 Prozent sicher sein, dass der Kunde damit zufrieden ist.

3.6 Risiko: Änderung der Anforderung

Es soll auch den Fall geben, dass ein Projekt zwar abgeschlossen wird, aber der Kunde es nicht mehr in dieser Form benötigt. Seine Anforderungen haben sich während des Projektablaufs so grundlegend geändert, dass das Produkt in dieser Form für ihn nicht mehr von Nutzen ist. Geht man davon aus, dass bei einem herkömmlichen Projekt Anforderungen geändert werden, so ist das im allgemeinen zwar realisierbar, aber mit sehr hohem Aufwand verbunden.

Wir können das zwar ändern, aber es kostet dann mehr und dauert etwas länger. XP versucht dies über seine kurzen Zyklen zu vermeiden. Durch Einbinden des Kunden in das Projekt wird es laufend den Wünschen des Auftraggebers angepasst (Stichwort Kommunikation). Weitreichende Anforderungsänderungen sind dadurch wesentlich unwahrscheinlicher. Hier geht man eher von der Grundeinstellung aus, dass Änderung die einzige Konstante ist, die auftreten kann. Es gibt keinen Unterschied zwischen Anforderungsänderungen und sonstigen Änderungen. Und man kann sehr schnell auf Anforderungsänderungen reagieren. Man erwartet sogar vom Kunden dass er seine Anforderungsdefinitionen jederzeit überdenkt.

3.7 Risiko: Featuritis

Wer kennt das nicht aus vielen Programmen, die so umfangreich an Features sind, dass man sie

- a: nicht überschauen kann und deshalb nicht nutzt
- b: nicht brauchen kann

XP wirkt dem sehr einfach entgegen, indem es grundsätzlich auf seine Prioritätenliste achtet. Es werden nur Tasks mit höchster Priorität realisiert. Dadurch entgeht man auf sehr elegante Weise dem Problem zu viele Features einzubauen, die dann nicht benutzt werden oder überhaupt nicht benötigt werden.

4 Zeit/Kosten Faktor in XP

4.1 Wie teuer sind Änderungen?

- Traditionelle Sichtweise/Erfahrung

Man geht hier von der traditionellen Sichtweise aus. Man beginnt mit der Analyse der Anforderungen, schreitet zum Entwurf und schließlich zum Implementieren. Danach wird getestet und letztendlich geht das Projekt in Betrieb.

Es ist zu ersehen, dass umso mehr Zeit vergeht, Änderungen mit steigendem Kostenfaktor realisiert werden können. Dies zeigen auch Erfahrungen.

4.2 Wie teuer sind Änderungen?

- XP Sichtweise/Erfahrung(?)

Da XP noch sehr jung ist kann man hier nicht von Erfahrungswerten ausgehen. Trotzdem soll die Grafik verdeutlichen auf welches Ziel XP hinarbeitet. Durch die eher parallel gehaltenen Projektphasen wird versucht, Änderungswünschen auf einem kostengünstigen Weg entgegenzukommen. Änderungen sollen vor allem gegen Ende des Projektes keine Unsummen auslösen.

5 Grundwerte XP im Beispiel

5.1 Der Anfang allen Seins, die Problemschilderung

Jedes Projekt beginnt mit einem Auftrag. Bisher kam als nächstes ein Pflichtenheft mit den abzuliefernden Features. Anschließend war es Sache der Design-Gruppe, was zu welcher Zeit implementiert werden sollte. Bei XP formuliert man keine Anforderungen, sondern schildert Probleme, genannt "User Stories". Diese Beschreibungsform ist auch in der Unified Modeling Language (UML) gebräuchlich, dort heißt sie "Use Case".

Eine solche UserStory könnte im Beispiel eines Parkhausprojektes wie folgt aussehen:

Das Einfahren eines Fahrzeugs in ein Parkhaus:

- Fahrer bleibt an geschlossener Schranke stehen, fordert per Knopfdruck Parkschein an.
- Er entnimmt den Parkschein und daraufhin öffnet sich die Schranke.
- Fahrer passiert die Schranke.
- Schranke schließt wieder

XP stellt also keine Anforderungen wie in einem Pflichtenheft, sondern schildert die Probleme in Form von UserStories die auf UserCards geschrieben werden.

5.2 Die Umsetzung

Sind die meisten User Stories bekannt, beginnt das große Verhandeln. Kunde, Management und Entwickler legen fest, welche Stories als Erste zur Umsetzung kommen und bis wann die einzelnen Komponenten vorliegen sollen. Dabei kommen technische und kommerzielle Aspekte zum Tragen, weshalb Entwickler und Manager gemeinsam verhandeln. Es geben also nur die Ausführenden eine Terminabschätzung ab, denn Geschäftsleute neigen in solchen Sachen bekanntlich zu übertriebenem Optimismus. Nur wer die Arbeit letztlich macht, kann verlässliche Zeiten annähernd abschätzen - entsprechende Erfahrung vorausgesetzt.

Das Resultat dieser Phase ist ein "Release Plan". Dieser beinhaltet für jedes einzelne Ziel des Projekts:

- was zuerst implementiert werden muss, also die Reihenfolge der einzelnen User Stories,
- wie lange es dauert, dieses Ziel zu erarbeiten,
- Kosten für die Umsetzung.

Was wenn noch was fehlt?

Ein Aufmerksamer Beobachter könnte sich nun fragen: 'Was soll denn geschehen,

wenn der Parkscheinautomat keine Tickets mehr hat, oder das Parkhaus voll ist'? Auch könnte man sich an dieser Stelle fragen: 'Wissen eigentlich die Programmierer, der Kunde zur Release Planning oder auch zum Zeitpunkt, wenn eine Story kodiert wird, wie man z.B. eigentlich die dortigen Schranken ansteuern werden muss. Ein solches Detail ist aber von extremer Wichtigkeit für das Gelingen des gesamten Projektes. Deshalb sucht ein Mini-Team eine Prinziplösung, einfach um die verdeckten Probleme in dieser Detail-Aufgabe zu finden. Diese Lösung lässt sich zwar im Projekt nicht weiter verwenden, zeigt aber den Weg auf, ist also keine vertane Mühe. Neue Erkenntnisse werden einfach zu bestehenden UserStories hinzugefügt auf separaten UserCards. Je später ernste Probleme zu Tage treten, desto teurer kommen sie zu stehen. Die Verringerung derartiger Risiken ist ein Hauptanliegen von XP.

5.3 Die Codierung

Nun erst wird programmiert. Das Team weiß durch die gegebene Umsetzungsreihenfolge, die zuvor im Release-Plan festgelegt wurde, was zuerst funktionieren soll.

Als Erstes programmiert man immer den Test eines Modules, ebenso sorgsam wie die Funktionen der eigentlichen Anwendung. Die Testmodule entwickeln sich genauso wie das 'eigentliche' Programm über die gesamte Projektzeit weiter.

Erst nachdem der Test kodiert wurde beginnt das Kodieren der Module. Zunächst werden sie nur rudimentäre Grundfunktionen erfüllen - aber der Anfang ist gemacht.

Wenn die erste Iteration fertig ist, integriert man sie auch gleich. Das heißt aber nicht, alles Vorhandene einfach auf einen Haufen zu werfen und zusammenzubinden. Wenn dann nichts funktioniert, kennt man die Ursache nicht und muss sie durch Debugging erst suchen. Stattdessen integriert man die Module sequenziell: Auf einmal kommt immer nur eines zum Projekt hinzu. Treten dann Fehler auf, ist klar, woher die kommen.

5.4 Pair Programming

Man liest Zeile für Zeile eines Codes und trotz Kommentare bleibt die Idee des Ganzen verborgen. - wer kennt das nicht? Eigentlich kennt doch nur der Ersteller den Code, also seinen 'Geist'.

Wenn Sie sich nun einwerfen: "Sind denn keine Kommentare zum Code hinzugefügt worden?"

Sind wir mal ehrlich, wer würde ein Werk wie 'Die Räuber' aus einer Satzanalyse begreifen? Abhilfe sollte die Pflicht der Dokumentation schaffen, dort sollte man den Core des Codes finden, oft ist aber gerade in Softwareprojekten die Zeit sehr knapp bemessen, denn Kodieren, Testen und Integrieren belegt die Entwicklermannschaft bereits schon vollständig.

Um diesen Mangel auszugleichen, sieht XP vor, immer zwei Programmierer an einem Arbeitsplatz vor. Zusammen schaffen die beiden eine Iteration einer Story. Sobald die Integration zu Ende ist, sucht sich einer des Zweierteams einen neuen Arbeitsplatz und der andere bekommt einen neuen hinzu, es wird also ständig *rochieren*. So soll am Ende jeder das ganze Projekt kennen und das ohne lange in Begleitdokumenten nachzulesen.

Jetzt schimpfen aber die Kaufleute - was das wieder kostet... - Das ist schon richtig,

andererseits, wenn zwei Menschen zusammen ein Problem lösen, hat jeder seine eigene Vorstellung, die vielleicht einen Fehler aufweist oder etwas nicht erfasst. Wenn zwei Gemüter an derselben Sache arbeiten debattieren sie und tauschen ihre Ideen aus. Schließlich wird etwas entstehen, das den Vorstellungen beider Entwickler entspricht. Natürlich könnten auch die Programmierer jammern: "Was, wenn der andere nicht so gut ist wie ich?" Dann wird der andere etwas lernen. Und er kann dennoch hier und da eine gute Idee haben. Und was ist mit zu schnell denkenden Leuten? Der "Schwächere" muss dann halt durch Fragen bremsen. Das bringt den Schnelldenker auch dazu, seine Ideen zu überprüfen - oder er ist vom "Greenhorn" genervt. Da es hier ganz erheblich um die Chemießweier Mitarbeiter geht, kann Pair Programming im Einzelfall auch schon mal mehr Probleme schaffen, als es löst.

Der geistige Vater des Extreme Programming, Kent Beck, erwähnt eher beiläufig, dass man zwar meist keine Dokumentation schreibe, es aber eigentlich tun sollte. Wohl auch, um diesen Mangel auszugleichen, sieht die Idee von XP immer zwei Programmierer an einem Arbeitsplatz vor.

Eine weitere Besonderheit des XP ist es, den Kunden während der gesamten Entwicklung mit einzubeziehen, der Idealfall wäre hier natürlich, dass z.B. ein Mitarbeiter des Kunden mit beim Entwicklerteam sitzt und somit für kurze Kommunikationswege sorgt, auch kann der Kunde nun sehr effizient dem Entwicklerteam die gewünschte Richtung vorgeben. - Defizite könne somit schnell entschärft werden.

5.5 Schrittweise Annäherung

In XP gilt die Regel: Tu nie mehr, als du tun musst! Je einfacher also desto besser?!'

Hierzu ein Beispiel, warum der Erfinder von XP soviel Wert auf die Einfachheit legt:

Für ein Softwareprojekt wurde ein allgemeines Dialogfeld gebraucht. Ein Programmierer war der Ansicht dieses Dialogfeld möglichst intelligent zu gestalten, also seine Größe und die Anzahl der Zeilenumbrüche im Text basierend auf der Schriftgröße und anderen Variablen festzulegen. Der Programmierer wurde gefragt, wer eine solch intelligente Schnittstelle bräuchte.

Es stellte sich heraus, daß nur dieser eine Programmierer diese Schnittstelle implementieren wollte, auch der Vorschlag das Dialogfeld etwas schlichter zu gestalten d.h. eine bestehende Klasse und Schnittstelle offen legen und diese genau nach den Anforderungen des Kunden anpassen, nicht mehr und nicht weniger als wirklich gefordert wurde (20 Minuten Arbeit).

Der Programmierer setzte sich durch es wurden zwei Tage mit dem Programmieren dieses Codes zugebracht.

Am dritten Tag hatten sich die Anforderungen geändert und man brauchte dieses Dialogfeld nicht mehr. Man hatte zwei Manntage in einem Projekt vergeudet, das ohnehin schon unter Termindruck stand.

Die entwickelte Software nähert sich schrittweise an das fertige Produkt an. Dadurch ruinieren Änderungen nicht gleich die ganze Planung und der Gesamtprozess bleibt wesentlich besser steuerbar. Niemand versucht, auf Anhieb eine Komplettlösung zu erzeugen: Wer weiß, wo noch Schwierigkeiten auftauchen und ob der Kunde nicht seine Wünsche noch ändert? Damit diese Annäherung funktioniert, muss man den Iterati-

onsplan ständig neu verhandeln - einmal die Woche sehen, wie weit man gekommen ist, sich fragen, ob die Schätzungen noch stimmen, und eingreifen, falls nicht.

Kam früher der Chef und fragte: "Wie weit seid ihr?", kam als Antwort vielleicht: "Das Design ist fast fertig". Nun gibt es greifbare Zahlen zur Antwort "Die User Story 'Einfahrtschranke' haben wir im Kastenöder "Es sind noch 15 von 37 Stories zu implementieren".

Durch die häufige Integration entstehen viele neue, getestete Versionen des Produkts, auf die sich der kundenseitige Abnahmetest anwenden lässt. Jedes Mal bei Erfolg bekommt der Kunde eine neue "Release". Das vermittelt dem Auftraggeber einen Eindruck vom Fortschritt.

Außerdem kann der Abnehmer die neue Release ausgiebig bei sich testen. Je früher eine neue Version fertiggestellt wird, desto früher kann der Kunde reagieren, Fehler und neue Wünsche zurückmelden, und umso weniger Zeit bleibt, darauf zu reagieren.

5.6 Grundwerte

Betrachtet man XP nun näher so kann man feststellen, dass stets 4 Grundwerte eine bedeutende Rolle spielen.

1. Kommunikation
2. Einfachheit
3. Feedback
4. Mut

Kommunikation und Einfachheit

- Zwischen Einfachheit und Kommunikation besteht eine wechselseitig Beziehung. Je mehr man kommuniziert, desto klarer erkennt man, was wirklich getan werden muss. Je einfacher ein System ist, desto weniger muss darüber mitgeteilt werden, was zu einer umfassenderen Kommunikation führt.

Kommunikation, Einfachheit, Feedback und Mut

- Kommunikation fördert Mut, da damit Möglichkeiten für risikoreichere, lohnenswerte Experimente eröffnet werden. "Dir gefällt das nicht? Ich hasse diesen Code. Lass uns einmal ausprobieren, wie viel wir davon in einem Nachmittag ersetzen können."
- Einfachheit fördert Mut, da man viel mutiger sein kann, wenn man mit einem einfachen System zu tun hat. Es ist viel weniger wahrscheinlich, dass man ein solches System versehentlich beschädigt.

- Mut fördert wiederum Einfachheit, da man das System zu vereinfachen versucht, sobald man eine Möglichkeit hierzu sieht.
- Konkrete Feedbacks fördern Mut, da man sich viel sicherer fühlt, wenn man nach einer radikalen Codeänderung eine Taste drücken kann und sieht, dass die Tests funktionieren (oder nicht, in welchem Fall Sie den Code wegwerfen).

6 Die Rollenverteilung -Die Spieler in XP-

6.1 Der Truckfaktor

Der Truck-Faktor gibt an, mit welcher Wahrscheinlichkeit das Projekt scheitern wird, wenn ein Teammitglied von einem Truck überfahren wird. Der höchste Truck-Faktor (1.0) wird erreicht, wenn das Team aus Spezialisten besteht, von denen jeder seinen Teil des Systems perfekt kennt. Der niedrigste Truck-Faktor (0.0) wird erreicht, wenn "Collective Code Ownership" praktiziert wird.

6.2 Rollen

6.2.1 Auftraggeber (Client)

- Für zentrale Entscheidungen zuständig
- Definiert die Ziele (Umfang)
- Stellt finanzielle Mittel bereit
- Stellt die Zielerreichung fest
- Damit hängt der Erfolg eines Projektes wesentlich vom Auftraggeber ab
- Nimmt am Planungsspiel für Projektetappen (Releases) teil

6.2.2 Kunde (Customer)

- Der Kunde legt fest, was zu tun ist (Prioritäten)
- Dazu schreibt er die User Stories
- Diese müssen verständlich, umsetzbar, testbar sein
- Schreibt zusätzlich die funktionalen Abnahmetests und führt diese, mit Hilfe des Testers, durch
- Diese funktionalen Tests sagen ihm, ob das System das tut, was er erwartet
- Als Mitglied des Teams nimmt er an den Planungsmeetings (Release und Iteration) und den täglichen Stand-up Meetings teil
- Er ist für aufkommende Fragen bezüglich der Stories immer ansprechbar
- Der Kunde beeinflusst die Entwickler durch permanentes Feedback
- Dazu muss er während der Entwicklung immer vor Ort sein

6.2.3 Programmierer (Programmer)

- Die tägliche Arbeit besteht aus Zuhören, Testen, Implementieren und Design
- Die Programmierer haben einen großen Anteil bei der Planung, da nicht das Management die Pläne vorgibt, sondern das Team diese ausarbeitet
- Ehrlichkeit ist wichtig ->Man muss sich über seine Stärken und Schwächen im klaren sein und um Hilfe bitten können ->Außerdem funktioniert die Aufwandsabschätzung nur mit Ehrlichkeit (und Erfahrung)
- Programmierer sollen gut sein, aber "Super-Coder" sind unnötig (können sogar behindern)
- Teamfähigkeit ist wichtig, denn es wird ständig mit wechselnden Paaren gearbeitet und der Quelltext gehört dem Team
- Andere Skills wie die Fähigkeit zum "Refactoring" und zum "Unit Testing" werden (zwangsläufig) in XP gelernt werden

6.2.4 Tester

- Der "Tester" hilft dem Kunden bei der Erstellung und Durchführung der funktionalen Tests
- Außerdem sammelt er die Ergebnisse und macht sie für alle sichtbar (Qualität)
- Die Rolle wird normalerweise von den Entwicklern oder vom Coach übernommen

6.2.5 Verfolger (Tracker)

- Der "Tracker" ist das Team Gewissen
- Gibt dem Team feedback wie gut die Wirklichkeit mit der Abschätzung übereinstimmt und weist das Team auf mögliche Probleme hin
- Er ermittelt regelmäßig (z.B. nach Abschluß einer Aufgabe) die Zeit die für die Aufgabe nötig war und aktualisiert den "Load Faktor"
- Er beobachtet den Entwicklungsprozess und überwacht ob die Pläne eingehalten werden können
- Die Rolle wird normalerweise von den Entwicklern oder vom Coach übernommen

6.2.6 Trainer (Coach)

- Der Coach ist für das Funktionieren des Prozesses verantwortlich. Dazu benötigt er einen guten Überblick über das System
- Wenn etwas schief läuft, muss der Coach wieder in die richtige Richtung lenken, möglichst aber indirekt (Selbstverantwortung der Entwickler)
- Außerdem sollte er technische Fähigkeiten haben, um z.B. als Programmierpartner besonders für neue Teammitglieder verfügbar zu sein

- Außerdem sollte er technische Fähigkeiten haben, um z.B. als Programmierpartner besonders für neue Teammitglieder verfügbar zu sein

7 Der Weg zu einer Lösung

7.1 Einflußfaktoren

- Kostenrahmen
- Zeitrahmen
- Qualität
- Funktions-Umfang

7.2 Einfluß des Kostenrahmens

- zu wenig Geld
 - keine effektive Entwicklung
- mehr Geld
 - bessere Umgebung, Ausstattung, Ausbildung,.....
- aber Geld allein macht kein erfolgreiches Projekt
 - "40 Programmierer"-Anekdote
- besser
 - Projektgröße schrittweise anpassen
- Problem
 - Status-/ Prestige-Denken von Projektleitern

7.3 Einfluss des Zeitrahmens

- zu wenig Zeit
 - schlechter Qualität
 - geringer Umfang
 - hohe Kosten
- mehr Zeit
 - bessere Qualität
 - mehr Funktionalität
- aber zu viel Zeit bis Inbetriebnahme schadet
 - Kein Feedback aus laufendem Betrieb
- wenig Einflussmöglichkeiten durch Programmier-Team
 - Zeitrahmen ist meist vom Kunden bestimmt

7.4 Einfluss der Qualität

- externe Qualität
 - was der Kunde sieht
- interne Qualität
 - was der Programmierer sieht
- geringere interne Qualität
 - was der Programmierer sieht
 - kurzfristige Zeitersparnis
 - langfristige Wartbarkeitskatastrophe
 - langfristig schlechte externe Qualität
 - demoralisierender Effekt im Team
- hohe interne Qualität
 - langfristig schnellere Entwicklung
 - bessere Motivation/ Zufriedenheit des Teams
 - Kundenzufriedenheit

7.5 Einfluss des Funktions-Umfangs

- Wichtigste Einflussmöglichkeit (laut Kent Beck)
- Idee
 - Kosten, Zeit und Qualität festlegen
 - realisierbaren Funktionsumfang
- Vorteile
 - Leichte Anpassbarkeit an Änderungsanforderungen
- Risiko
 - Zu viele/ zu wichtige Funktionen werden gestrichen
- XP-Ansatz
 - wichtigste Kundenanforderungen zuerst realisieren
 - Aufwandschätzungen mit Feedback

7.6 Einflussfaktoren: Fazit/Empfehlungen

- Kosten: Ressourcen bedarfsgerecht einsetzen!
- Zeit : keine Einflussmöglichkeit , da extern bestimmt !
- Qualität : hohe interne Qualität anstreben!(Bestimmt durch Kunden)
- Umfang : minimalen Funktions-Umfang anstreben!(Bestimmt durch Projektleiter)

8 Wann ist XP nicht sinnvoll?

Im Vordergrund steht, dass das Team sich versteht. Es ist für kein Projekt von Vorteil wenn sich einzelne Teammitglieder aus dem Weg gehen weil sie sich nicht "riechen" können. Gerade bei XP, wo das Verstehen untereinander sehr auf die Probe gestellt wird über das Pair Programming, sollte im Vorfeld eine Homogene Teammitgliederauswahl getroffen werden.

XP sollte nicht eingesetzt werden wenn das Management oder der Kunde klare Richtlinien vorgeben oder das Management nicht bereit ist wichtige Entscheidungen den Programmierer zu überlassen, und sozusagen dem Team nicht die Führung übergeben. Die Projektgröße spielt natürlich auch eine Rolle. Sollte das Projekt Spezialisten benötigen kann XP nicht korrekterweise angewandt werden. (Keine Ergänzung und Wissensbereicherung für andere Teammitglieder!). Die Teamgröße sollte nicht über 10 bis 12 Programmierer hinausgehen. Dementsprechend sollte auch die Projektgröße mit dieser Anzahl zu Realisieren sein.

Auch ist es nicht sinnvoll XP einzusetzen wenn der Kunde nicht am Projekt integriert werden kann, aufgrund von örtlichen Begebenheiten. (Z.B. Kunde in Australien, Entwicklerfirma in Frankreich). Wenn es nicht möglich ist den Kunden einzubinden in das Projekt, macht es keinen Sinn XP anzuwenden, denn der wesentliche Bestandteil, der Kunde, ist maßgebend an der Entwicklung beteiligt.

...

Das wichtigste ist kurz und salopp gesagt: Das Team an sich muss sich gut verstehen und ergänzen. Und zu viele Köche verderben den Brei. Und vorallem: "DER KUNDE IST KÖNIG"

9 Quellennachweis

- <http://www.xprogramming.com>
- <http://www.xpdeveloper.com>
- <http://www.extremeprogramming.org>
- <http://www.pairprogramming.org>
- <http://www.refactoring.com>
- Kent Beck, eXtreme Programming; Addison-Wesley, 2000; ISBN: 3-8273-1709-6

10 Anhang

In dieser Ausarbeitung sind eine Vielzahl von Präsentationen, Vorträge, Beiträge verschiedenster Personen eingeflossen, welche teilweise überarbeitet oder zusammengefasst wiedergegeben werden. Alle Rechte bleiben bei den ursprünglichen Eigentümern dieser Rechte. Keines dieser Rechte wird von uns direkt oder indirekt übernommen, selbstverständlich auch dann nicht wenn übernommene Passagen nicht als solche gekennzeichnet sind.