Seminar Software Engineering
Universität Zürich, Winter 03/04

Agile vs. klassische Methoden der Software-Entwicklung

eXTREME PROGRAMMING

Manuel Meyer
Rosenstrasse 9, 8152 Glattbrugg
Matrikelnr. 99-905-739

Institut für Informatik der Universität Zürich

Dozent: Prof. Dr. M. Glinz
Betreuer: C. Seybold

Inhaltsverzeichnis

1. Grundlagen

1.1 Was ist XP?

XP ist eine Methodik zur Entwicklung von Software, die sich durch folgende Eigenschaften auszeichnet:

Simpel
Effizient
Tiefes Risiko
Flexibel
Voraussagbar
Wissenschaftlich
Nach XP Methodik zu arbeiten soll Spass machen!

1.2 Woher kommt XP?

In den frühen 1990er Jahren hat sich ein Softwareentwickler namens Kent Beck mit der Frage beschäftigt, wie man die Entwicklung von Software verbessern könnte. Zusammen mit Ward Cunningham entwickelte er eine Strategie, die Softwareentwicklung einfacher und zugleich effizienter machen sollte. Im März 1996 startete er ein Projekt für Daimler-Chrysler, in welchem er seine neu entdeckten Konzepte verwirklichte. Daraus entstand die Extreme Programming Methodik.

2. Extreme Programming Basics

2.1 Warum braucht es XP?

2.1.1 Risiko

Das Hauptproblem bei der Entwicklung von Software ist das Risiko. Das Risiko, dass Software nicht zur Zeit geliefert werden kann oder dass Software für den Kunden keinen Wert erzeugt. Diese Probleme haben starke Konsequenzen, welche über Abbruch oder Weiterführung eines Projektes bestimmen können. Es muss ein Ansatz gefunden werden, mit welchem diese Risiken und deren ökonomischen und menschlichen Auswirkungen minimiert werden können.

Die folgenden Beispiele zeigen einige der Risiken auf, welche bei der Entwicklung von Software auftreten und wie Extreme Programming damit umgeht:

Zeitpläne
Problem: Der Tag der Lieferung naht und der Kunde muss informiert werden, dass die Software noch nicht geliefert werden kann.

Extreme Programming:
Extreme Programming beruht auf sehr kurzen Releasezyklen, welche höchstens ein paar Monate beinhalten. So wird die Gefahr eingedämmt, dass wegen zu langen Zeitabschnitten die Planung ausser Kontrolle gerät. Weiter werden die Zeitabschnitte in 1

bis 4 wöchige Iterationen unterteilt, in welchen kundenbezogene Features entwickelt werden und durch sofortiges Feedback auf ihre Funktion überprüft werden. Dazu kommt, dass die Aufgabenverteilung in Extreme Programming auf Ein- bis Dreitagesbasis geplant wird, damit das Team auch innerhalb einer Iteration Probleme lösen kann. Ferner werden grundsätzlich immer die Features mit der höchsten Priorität zuerst implementiert. Dies hat zur Folge, dass Features, die es nicht in den aktuellen Release schaffen, tieferen Wert erzeugen.

Projekt abgebrochen
Problem: Das Projekt wird nach mehreren Iterationen abgebrochen, ohne je in die Produktion zu gelangen.

Extreme Programming:
Der kleinste Release, der am meisten Sinn für den Kunden macht, wird von diesem bestimmt. Wegen der kleinen Releasegrösse kann weniger schief gehen, bevor mit der Produktion begonnen wird. Da der Kundenvertreter die Prioritäten festlegt, ist der Wert der produzierten Software maximal.

Die Software wird unbrauchbar
Problem: Das System ist produziert und in Betrieb. Nach wenigen Jahren werden die Kosten zur Änderung oder die Anzahl Fehler so gross, dass das komplette System ersetzt werden muss.

Extreme Programming:
Es wird ein grosser Bestand an Tests gehalten, welche nach jeder Änderung automatisch, bei Bedarf sogar mehrmals am Tag, durchgeführt werden. Das System wird stets in bester Verfassung gehalten. Überflüssiger Code wird sofort entfernt. Sogar die Architektur des Systems darf unter bestimmten Voraussetzungen kurzfristig geändert werden.

Fehlerrate
Problem: Das System ist produziert, die Fehlerrate ist aber so hoch, dass es nicht benutzt wird.

Extreme Programming:
Durch die vielen Tests wird die Fehlerrate minimal gehalten. Es wird unterschieden zwischen Funktionstests, welche von den Programmierern geschrieben werden und Programm-Feature Tests, welche von dem Kunden geschrieben werden.

Missverständnis zwischen Kunde und Entwicklern
Problem: Das System ist produziert, löst aber nicht das Problem des Kunden.

Extreme Programming:
Der Kunde ist während des ganzen Projekts ein Mitglied des Entwicklungsteams. Die Spezifikation der Software wird mit seiner Hilfe stets verfeinert.

Vergoldung
Die Software ist zwar reich an interessanten Features, bringt dem Kunden aber nichts.

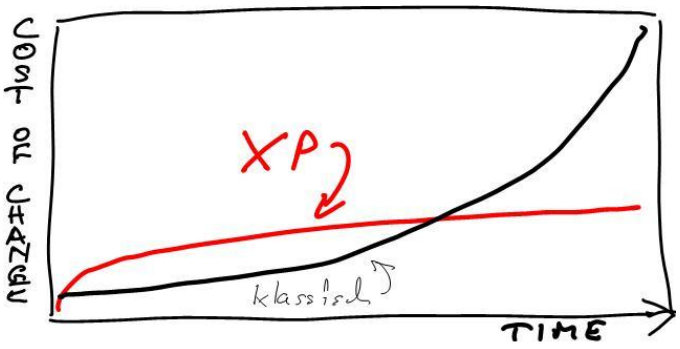
Extreme Programming:
Es werden immer die vom Kunden als Wichtigste angesehenen Features implementiert, und er hat die Möglichkeit zu intervenieren, wenn dies nicht der Fall wäre.

Personalwechsel
Nach 2 Jahren hassen die Programmierer das Projekt und gehen.

Extreme Programming:
Die Teamkultur ist in XP enorm wichtig. Kommunikation wird gross geschrieben und das Programmieren soll Spass machen. Jedes Mitglied muss Verantwortung übernehmen und bekommt Feedback über seine Arbeit. Zufriedenheit der Mitarbeiter steht im Vordergrund.

2.1.2 Kosten

Eine der Kernaussagen in der klassischen Softwareentwicklung ist, dass die Kosten, die zur Änderung der Software nötig sind, exponentiell steigen im Verhältnis zur Projektlaufzeit. Extreme Programming nimmt an, dass es möglich ist, durch bestimmte Massnahmen die Kostenkurve zu verändern. Das Ziel ist, dass sie viel flacher verläuft und sich im besten Fall sogar einer Horizontalen annähert.



Tatsächlich gibt es Technologien und Praktiken, welche es ermöglichen, den Verlauf der Kostenkurve zu verändern.

Zum Beispiel sind Objekte und Nachrichten Schlüsseltechnologien, wenn es darum geht, einen leicht änderbaren Code zu erzeugen. Jede Nachricht ist ein potentieller Punkt, an welchem man Veränderungen am Programm vornehmen kann, ohne dass man den Code grundlegend ändern muss.

Folgende Punkte sind massgebend, um Code zu erzeugen, den man ohne zu hohe Kosten ändern kann:

- **Einfaches Design.**
  Keine Ideen implementieren, welche man noch nicht braucht, aber in Zukunft brauchen möchte.

- **Automatische Tests**
  Das Verhalten des Systems kann ohne zusätzlichen Aufwand und in minimaler Zeit überprüft werden.

- **Stetige Anpassung des Designs**
  Durch stetige Anpassungen des Designs wird die Hemmschwelle, das Design zu ändern, abgebaut, und die Entwickler lernen, wie sie das Design effizient ändern können.

Mit der Idee im Kopf, dass wir die Kostenkurve flacher biegen können, müssen Annahmen der klassischen Software Entwicklung ganz anders betrachtet werden.

Schwerwiegende Entscheidungen können nun so spät wie möglich gemacht werden, da die Kosten zur Änderung nicht so schnell steigen. Je später wir Entscheidungen fällen können, desto grösser ist die Wahrscheinlichkeit, dass wir sie richtig fällen. Natürlich muss jede Änderung durch die automatischen Tests überprüft werden. Wir implementieren nur, was wir unbedingt benötigen, da wir zusätzliche Features ohne Probleme später implementieren können. Später werden wir eine genauere Idee davon haben, was wir brauchen und deshalb schneller zur richtigen Lösung kommen. Somit vermeiden wir Kosten, die durch implementierte Features entstehen, die wir gar nicht benutzen.

Designelemente werden nur eingeführt, wenn sie den bestehenden Code einfacher machen oder den nächsten Schritt vereinfachen, da wir das Design auch später verändern können.

Eine der Grundideen ist, das Extreme Programming mit Veränderungen umgehen kann. Radikale Änderungen sind möglich. Sie werden aber vermieden, da während dem ganzen Projekt die Richtung in kleinen Schritten zum gewünschten Ziel gelenkt wird. Dies trägt auch der Tatsache Rechnung, dass sich das Ziel sehr wahrscheinlich im Verlauf des Projektes noch verändern wird. Es ist nicht so, dass man einen Weg bestimmt und ihn dann mit Gewalt bis zum Ende durchschreitet. Eher bestimmt man den nächsten Schritt, sobald man den letzten getan hat. Die Frage ist nicht, ob es Änderungen im Entwicklungsumfeld geben wird, sondern wie man damit umgehen kann. An diesem Punkt steht man bei den klassischen Softwareentwicklungsmethoden vor einem Problem.

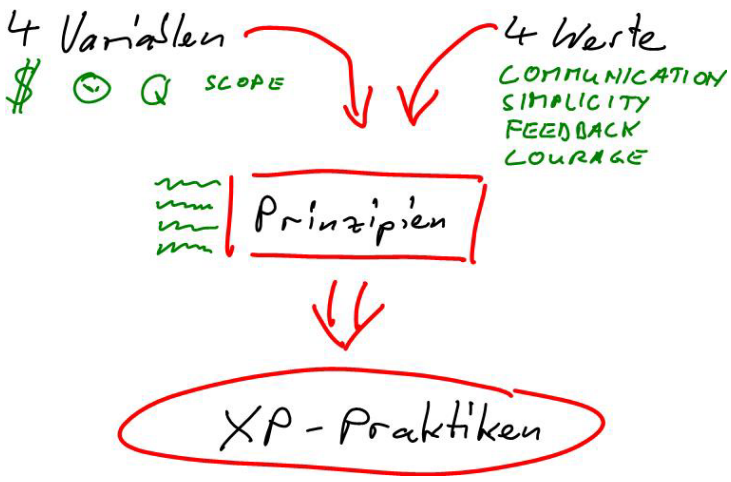
2.2 Wie funktioniert XP?

2.2.1 Grundsätze:

Extreme Programming ist mittlerweile schon ziemlich bekannt. Viele Informatiker haben bereits davon gehört. Die folgenden 4 Grundsätze von XP sind meist bekannt:

- **Pair-Programming**
- **Testgesteuerte Entwicklung**
- **Programmierer entwickeln das System, nicht nur einzelne Bestandteile**
- **Integration folgt unmittelbar auf Entwicklung**

Kent Beck beschreibt XP mit einem Modell, das 4 Variablen benützt; Diese sind Kosten, Zeit, Qualität und Umfang (Scope). Weiter beschreibt er 4 Werte, welche angeben, in welche Richtung sich das Projekt bewegt. Die Werte sind: Kommunikation, Einfachheit, Feedback und Mut. Mit Hilfe dieser Variablen und Werten werden dann Prinzipien hergeleitet, die grundlegend sind für ein XP Projekt. Schlussendlich leitet er aus den Prinzipien dann Praktiken her, welche es umzusetzen gilt.



2.2.2 Die 4 Variablen

Kent Beck beschreibt Extreme Programming mit einem Modell, das vier Variablen benützt: Kosten, Zeit, Qualität und Umfang (Scope).

Er beschreibt das Modell so, dass externe Kräfte, wie das Management oder die Kunden, drei der vier Variablen festsetzen und das Entwicklungsteam den Wert der vierten Variable festlegt.

Beziehungen zwischen den 4 Variablen

Kosten:
Mit zu wenig Geld kann das Problem des Kunden nicht gelöst werden. Zuviel Geld ergibt häufiger mehr Probleme als es löst. Es gilt, die richtige Menge zu finden. Für Manager sind die Kosten oft die wichtigste Variable überhaupt. Oft werden die Kosten auch mit Status verbunden, was zu ineffizienten Projekten führt. Zum Beispiel durch Aussagen wie: „Ich leite ein Projekt mit 100 Programmierern *protz*", obwohl 10 Programmierer die Arbeit schneller und besser ausführen könnten.

Zeit:
Mehr Zeit kann die Qualität verbessern und den Umfang vergrössern. Zu viel Zeit ist schlecht für ein Projekt, da wertvolle Erfahrungen gesammelt werden, wenn die Software im Betrieb ist. Dies wird durch zu viel Zeit hinausgezögert. Zu wenig Zeit hat schlechtere Qualität und somit höhere Kosten zur Folge. Die Zeit wird meistens von der Seite des Kunden vorgegeben und kann vom Entwicklungsteam nicht gross beeinflusst werden.

Qualität:
Durch absichtliche Opferung der Qualität können kurzfristig Probleme gelöst werden, langfristig sind die Schäden aber enorm. Qualität ist die Variable, die sich am speziellsten verhält. Einerseits kann mit zu vielen Massnahmen zur Sicherung der Qualität das Projekt träge und langsam werden, andererseits kann das Projekt durch gezielte Massnahmen auch schneller werden, wie zum Beispiel durch Codierrichtlinien oder Tests. Häufig kann Qualität auch zur Motivation für die Mitarbeiter werden, da sie sehen, dass sie gute Arbeit leisten.

Umfang:
Ein kleinerer Umfang macht es möglich, bessere Qualität zu liefern. Ist der Umfang zu klein, wird das Problem des Kunden nicht gelöst. Ein zu grosser Umfang schlägt sich in zu hohen Kosten nieder.

Der Umfang ist die wichtigste Variable im Extreme Programming. Häufig sind die Anforderungen der Kunden unklar und sie wissen erst, wenn der erste Release läuft, was sie im zweiten wollen oder im ersten eigentlich gewollt hätten. Der Umfang ist eine „weiche" Variable, die den sich ändernden Verhältnissen schnell angepasst werden kann. Somit kann man Kosten, Zeit und Qualität festlegen und den Umfang kontinuierlich danach richten.

2.2.3 Die 4 Werte

Bevor wir die Praktiken der Softwareentwicklung mit XP genauer anschauen, brauchen wir ein Mass, das uns zeigt, ob wir uns in die richtige Richtung bewegen. In XP gibt es 4 Werte, die uns zeigen, ob wir uns auf ein gemeinsames Ziel bewegen oder nur eigene Interessen verfolgen.

Diese Werte sind: Kommunikation, Einfachheit, Feedback und Mut.

Kommunikation
Die meisten Fehler, die in Softwareprojekten passieren, lassen sich auf mangelnde Kommunikation zurückführen. XP versucht, die Kommunikation aufrecht zu erhalten, indem es viele Praktiken verwendet, welche Kommunikation erzwingen. Zu diesen zählen Pair-Programming oder Zeitabschätzungen. Viel Kommunikation geschieht über Code. Code zeigt viel deutlicher, wovon man spricht, als natürliche Sprache. Der Code ist zentrales Element der Kommunikation.

Einfachheit
In XP muss man sich auf das konzentrieren, was man jetzt gerade machen muss, ohne in die Zukunft zu schauen. Grundsätzlich geht man davon aus, dass man nur implementiert, was man gerade braucht. Alles andere kann später gemacht werden. Auch bei den Programmiertechniken sollte man stets betrachten, ob es nicht eine einfachere Lösung gibt, die den gleichen Zweck erfüllt. XP geht davon aus, dass es sich lohnt, heute eine einfache Lösung zu benutzen, dies gilt selbst, wenn man morgen dafür etwas mehr Aufwand haben wird. Meist ist nicht klar, ob man diese Idee überhaupt noch weiter verfolgt.

Kommunikation und Einfachheit hängen sehr stark zusammen. Je mehr man kommuniziert, desto klarer sieht man, was nötig ist und desto einfachere Lösungen kann man entwickeln. Andererseits ist weniger Kommunikation nötig, wenn das System simpel ist und man die Kommunikation auf wesentliche Aspekte beschränken kann.

Feedback
Feedback ist enorm wichtig. Feedback geschieht parallel auf mehreren Ebenen. So bekommt der Entwickler durch die Modultests ein Feedback, ob der Code, den er gerade geschrieben hat, fehlerfrei funktioniert. Weiter wird das Team fortlaufend informiert, wie der Zeitplan aussieht. Auch der Kunde bekommt ein Feedback über den Projektfortschritt und erkennt anhand der funktionalen Tests, wie gut die Software bereits funktioniert.

Kontinuierliches Feedback geht einher mit Kommunikation und Einfachheit. Je mehr Feedback man bekommt, desto einfacher ist Kommunikation, und je simpler das System ist, desto einfacher ist es zu testen.

Mut
Der schwierigste Wert, den es in XP zu erhalten gilt, ist Mut. Man muss den Mut haben, unnötig komplizierten Code wegzuschmeissen und komplett neu zu beginnen. Dies ist

auch eine der Ideen von XP. Stellst Du am Abend fest, dass der Code, den Du entwickelt hast, nicht richtig funktioniert, wirf in weg und beginne am nächsten Tag von neuem. Weiter braucht man den Mut, einschneidende Veränderungen an der Architektur während dem Projekt durchzuführen, vorausgesetzt sie tragen zur Vereinfachung des Systems bei. Als Konsequenz solcher Eingriffe wird man mit der Tatsache konfrontiert sein, dass vielleicht die Hälfte aller Modultests nicht mehr funktioniert. Dann hat man die harte Arbeit, sich einen Test nach dem anderen vorzunehmen bis alle wieder richtig funktionieren. Später wird man aber damit belohnt, dass man eine einfachere Architektur als Grundlage hat.

Kent Beck beschreibt in seinem Buch mehrere Fälle, in welchen ein Mitglied des Teams eine radikale Idee hatte, um das Design zu ändern. Dem Mut des Teams war es zu verdanken, dass diese Idee umgesetzt wurde. Später stellte sich heraus, dass sich völlig neue Perspektiven ergaben, an die vorher niemand gedacht hätte.

Das Feedback hat hier einen sehr starken Einfluss auf den Mut. Wenn man die Möglichkeit hat, die Funktionalität des Systems anhand der Tests in ein Paar wenigen Minuten zu überprüfen, ist man nicht so abgeneigt, Änderungen am bestehenden Code vorzunehmen, was wiederum zu einem einfacheren System führt.

2.2.4 Die wichtigsten Prinzipien von XP

Die vier Werte Kommunikation, Einfachheit, Feedback und Mut geben uns Kriterien für erfolgreiche Lösungen von Problemen. Aus ihnen können wir nun die wichtigsten Prinzipien von XP herleiten. Diese Prinzipien geben uns die Möglichkeit, zwischen vorhandenen Alternativen auszuwählen. Sie reflektieren die oben genannten Werte, sind aber viel genereller anwendbar, da sie nicht von subjektiven Auslegungen beeinträchtigt werden.

Es gibt folgende 5 Hauptprinzipien in XP:

Unmittelbares Feedback (rapid feedback)
Die Psychologie lehrt uns, dass es einen starken Zusammenhang zwischen dem Lernerfolg und der Zeit gibt, die zwischen Aktion und Feedback verstreicht. Darum legt XP sehr viel Wert auf Feedback in verschiedenen Zeitebenen. Der Programmierer bekommt innerhalb von Sekunden ein Feedback von seinem Programmierpartner, da dieser ihn stets überwacht. Innerhalb von Minuten bekommen die Programmierer ein Feedback, indem sie die automatischen Tests laufen lassen. Am Ende des Tages bekommen sie ein Feedback, wenn sie den entwickelten Code ins bestehende System integrieren. Natürlich bekommen sie auch für jeden Releasezyklus und für den gesamten Projektfortschritt ein Feedback vom Projektleiter.

Annahme der Einfachheit (assume simplicity)
„Nehme an, dass jedes Problem auf lächerlich einfache Weise lösbar ist. Löse das Problem, so wie es sich Dir heute stellt, denke nicht an Morgen."

Im Gegensatz zu klassischen Entwicklungsstilen entwickeln wir die Software nicht mit Blick in die Zukunft und denken nicht an Erweiterungen oder Wiederverwendbarkeit. In XP muss man sich zutrauen, Funktionalität, die man später noch hinzufügen will, später implementieren zu können.

Inkrementelle Weiterentwicklung (incremental change)
Grosse Veränderungen auf einen Schlag funktionieren nicht. In XP wird im Allgemeinen jede Anpassung kontinuierlich und in kleinen Schritten durchgeführt. Der Plan ändert ein wenig, das Design ändert ein wenig, das Team ändert ein wenig, etc.

Änderungen willkommen heissen (embrace change)
„Die beste Strategie ist diejenige, welche die meisten Optionen offen lässt und zugleich die dringendsten Probleme löst." Änderungen sind willkommen, weil man gelernt hat, damit umzugehen.

Qualitätsarbeit leisten (quality work)
Jeder Mitarbeiter möchte gute Arbeit verrichten und an einem Projekt arbeiten, das Erfolg hat. Qualität ist wichtig für die Motivation.

Dazu kommen folgende weniger zentralen Prinzipien:

Lernen lehren (teach learning)
Anstatt genaue Vorgaben zu machen, legt XP Wert darauf, dass die Mitarbeiter selbst lernen, wie sie arbeiten sollen. Sie sollen selbst lernen, wie viele Tests nötig sind oder wie sie ein einfaches Design entwickeln können. Dies trägt auch dazu bei, dass sie ihre Arbeitsweise stetig verbessern. Unmittelbares Feedback unterstützt diese Art des Lernens natürlich.

Kleine Startinvestition (small initial investment)
Ein grosses Budget am Anfang des Projektes ist verheerend. Wenn das Budget knapp ist, muss das Team vollen Einsatz zeigen und die richtigen wegweisenden Entscheidungen treffen. Natürlich darf das Budget auch nicht allzu knapp sein.

Play to win (DON'T play not to lose)
Das Team muss von seinen Stärken überzeugt sein, und einen guten Zusammenhalt haben.

Experimente (concrete experiments)
Immer wenn man eine Entscheidung fällt und sie nicht ausprobiert, ist die Wahrscheinlichkeit gross, dass sie falsch ist. Deshalb ist es wichtig, dass man Entscheidungen mit Experimenten stützt.

Offene, ehrliche Kommunikation (open, honest communication)
Kommunikation muss offen und ehrlich sein. Unterschiedliche Meinungen müssen diskutiert werden, und das Team muss auch schlechte Nachrichten dem Management mitteilen können.

Mit dem Instinkt der Mitarbeiter arbeiten, nicht dagegen
XP funktioniert nur, wenn die langfristigen Ziele des Projektes auch dem kurzfristigen Instinkt und den kurzfristigen Interessen der Mitarbeiter entsprechen. Kurzfristig kann man Mitarbeiter „zwingen" etwas zu tun, langfristig führt das jedoch nicht zum Erfolg.

Verantwortung akzeptieren (accepted responsibility)
In XP wird Verantwortung akzeptiert und nicht gegeben. Nichts ist frustrierender, als gesagt zu bekommen, was man zu tun hat. Vor allem, wenn man das Gefühl hat, dass es sich um ein unlösbares Problem handelt.

Lokale Adaption (local adaption)
In einem Projekt muss man all sein Wissen über XP den lokalen Gegebenheiten anpassen. XP ist kein Rezept, dass immer funktioniert, eher eine Anweisung wie man seine eigene Methodik entwickelt. Jeden Tag muss man sich fragen, ob das, was man

heute gemacht hat, morgen auch noch funktionieren wird oder ob man etwas ändern muss. Kent Beck hat das sehr treffend formuliert:

„Don't read this (sein Buch) thinking, „Finally, now I know how to develop (Software)." You should end up saying, "I have to decide all of this AND program?" Yes you do. But it's worth it!"

Reisen mit leichtem Gepäck (travel light)
Kent Beck nennt die XP Entwickler „intellektuelle Nomaden". Das einzige Gepäck, was sie dabei haben, sind Tests und Code. Sie sind jederzeit bereit, in eine neue Richtung aufzubrechen. Sei es, dass das Design in eine andere Richtung geht, der Kunde eine andere Richtung einschlagen will oder sich das Team oder die Technologie ändert.

Ehrliche Messungen (honest measurement)
Messungen müssen sinnvoll sein. „Etwa 2 Wochen" ist eine ehrlichere Messung, als „70,25 Stunden." Anzahl Codezeilen ist eine schlechte Messgrösse, wenn der Code kürzer wird durch einfachere Strukturen.

2.2.5 Die XP Praktiken

Ausgehend von den vier Werten und den Prinzipien kommen wir nun zur Praxis. Wir leiten 12 Praktiken her, die essentiell sind für die Entwicklung von Software mit Extreme Programming. Alle Praktiken sind relativ einfach. Sie ergänzen sich derart, dass die Schwächen der einen durch die Stärken der anderen ausgeglichen werden.

The Planning Game
Software Entwicklung ist immer ein Dialog zwischen dem, was man will und dem, was man kann. Der Umfang des nächsten Releases wird bestimmt, indem man die Wünsche des Kunden mit den Schätzungen der Techniker paart. Der Kunde bestimmt, was er in diesem Release verwirklicht haben will, die Prioritäten, die Zusammensetzung des Releases und die Daten. Die Techniker schätzen die nötigen Zeiten und Konsequenzen ab, schlagen Prozesse vor und entwerfen einen detaillierten Zeitplan.

Small Releases
Jeder Release sollte so klein wie möglich sein und zugleich den grössten Nutzen erzeugen.

Metapher
Jedes XP Projekt sollte von einer Metapher begleitet werden. Sie ist eine abstrakte Beschreibung des zu entwickelnden Systems, die den Beteiligten hilft, die grundlegenden Elemente des Projekts und ihre Beziehungen zu verstehen. Hinzu kommt, dass sie eine Terminologie festlegt, die während dem ganzen Projekt verwendet wird. Die Metapher ersetzt die Architektur, welche man in der klassischen Softwareentwicklung definieren würde.

Einfaches Design
Die richtige Wahl des Designs ist definiert durch vier Regeln:

1. Es muss alle Tests bestehen.
2. Es darf keine duplizierte Logik enthalten
3. Jede Absicht der Programmierer sollte verwirklicht sein
4. Es hat am wenigsten Klassen und Methoden

Einfaches Design wird erreicht, indem man alle Designelemente entfernt, die man entfernen kann ohne Regeln 1 bis 3 zu verletzen.

Testen
Tests werden grundsätzlich vor dem Code geschrieben. Man muss für alle Programmpunkte Tests schreiben, an welchen etwas schief gehen könnte. Entwickler schreiben Modultests, damit sie Vertrauen in das Programm bekommen, Kundenvertreter schreiben funktionale Tests, um zu kontrollieren, ob das Programm seinen Zweck erfüllt. Mit einer guten Sammlung an Tests kann man das Programm einfacher und schneller verändern.

Refactoring
Jeder Entwickler, der während dem Arbeiten einen Weg sieht, den bestehenden Code zu verbessern oder zu vereinfachen, sollte dies tun. So hat man zwar mehr Arbeit, um ein gewisses Feature zu implementieren, dafür ist es beim nächsten Feature weniger aufwändig und der Code bleibt einfach.

Pair Programming
Es sitzen 2 Programmierer an einem Computer. Der Eine schreibt und der Andere kontrolliert und überlegt sich, ob man das Problem einfacher lösen kann. Programmierer A konzentriert sich auf den Code, den er entwickelt. Programmierer B schaut zu und greift ein, wenn eine Stelle kommt, die er anders geschrieben hätte. Der Entwicklung des Codes wird vom Dialog der Entwickler begleitet. Programmierer B hat etwas mehr Übersicht und eine grössere Sichtweite. Er behält auch die (unmittelbare) Zukunft im Auge, überlegt sich, ob weitere Modultests nötig sind und ob der Ansatz gut gewählt worden ist. Die Paare werden immer wieder neu bestimmt (Jeder programmiert mit Jedem).

Collective Code Ownership
Jeder, der eine Möglichkeit sieht, den Code zu verbessern, sollte dies tun. Jeder ist verantwortlich für den gesamten Code. Jeder darf den gesamten Code verändern.

Kontinuierliche Integration
Unmittelbar nach der Entwicklung wird der Code integriert und so lange korrigiert, bis alle Tests wieder zu 100 Prozent fehlerfrei ablaufen.

40 Stunden Woche
Das Team muss jeden Morgen frisch und eifrig sein. Überzeit kann XP ins Aus treiben. Als Regel gilt: „Wurde in einer Woche Überzeit gearbeitet, darf es in der Nächsten keine Überzeit mehr geben."

On-Site Customer
Ein guter Kundenvertreter ist ein Teil des Teams. Die Firma, welche die Software in Auftrag gibt, muss einen ihrer Mitarbeiter während der ganzen Projektlaufzeit zur Verfügung stellen. Er muss sehr gut wissen, welche Anforderungen erfüllt sein sollten und über Weisungsbefugnis verfügen. Der Kundenvertreter steht dem Team während der ganzen Zeit zur Verfügung. Er kann jederzeit angesprochen werden, wenn Unklarheiten zum Vorschein kommen. Weiter ist er verantwortlich für die Funktionstests, welche testen, ob das Programm so reagiert, wie es gewünscht ist. Dies bedeutet aber nicht unbedingt, dass der Kundenvertreter ein Informatiker sein muss. Er kann die Testfälle mit einem Entwickler des Teams zusammen implementieren. Ausserdem hat der Kundenvertreter die Pflicht, einzugreifen, sobald das Projekt in eine ungewünschte Richtung zu laufen droht.

Coding Standards
Mit den Wechseln der Programmierpaare und dem stetigen Refactoring, muss man sich auf die Formatierung des Codes einigen, damit kein chaotischer Code entsteht. Bald kann man nicht mehr nachvollziehen, wer welchen Teil geschrieben hat.

2.2.6 Die XP Strategien

Im Folgenden werden wir Strategien auf verschiedenen Ebenen betrachten: Management, Planning, Development, Design und Testing.

Grundsätzlich geht es darum, dass eine Strategie gefunden wird, die sowohl die geschäftlichen Anforderungen (business), als auch die Technischen vereint. Häufig werden die Entscheidungen vom Business gefällt, gestützt von den Schätzungen und Inputs der Entwickler.

XP-Management Strategie

Die Kompetenzen zwischen Management und Team sind in XP nicht so stark geregelt. Wir berufen uns auf die oben erwähnten Prinzipien. Der Manager legt fest, was erledigt werden muss, nicht wer was macht (accepted responsibility). Das Verhältnis zwischen Team und Manager beruht auf Vertrauen und Respekt. Der Manager unterstützt die Teammitglieder, damit sie sich auf ihre Arbeit konzentrieren können (quality work). Der Manager gibt sanfte Richtlinien vor und nicht strenge Gesetze. Weiter passt er XP an die lokalen Gegebenheiten an (local adaption). Der Manager verzichtet so weit als möglich auf erschöpfende Meetings oder lange Reporte (travel light). Der Manager wählt passende Messgrössen um das Projekt zu verfolgen (honest measurement). Zudem kommt dem Manager eine Coaching- und Trackingfunktion zu. Das heisst, er kümmert sich um das Team und verfolgt den Projektfortschritt.

XP-Planning Strategie

Die Planung in einem Projekt beinhaltet vor allem folgende Punkte:

- Team zusammenbringen
- Umfang und Prioritäten wählen
- Kosten und Zeiten schätzen
- Vertrauen schaffen, dass das Projekt durchführbar ist
- Ein Mittel für kontinuierliches Feedback bereitstellen

Grundsätzlich wird die Planung in XP nur auf eine kurze Dauer gemacht. Man plant nur bis zum nächstem Horizont, das heisst bis zum nächsten Release oder der nächsten Iteration. Eine Langzeitplanung existiert nur grob. Die Planung wird stets angepasst. Man will nicht Ressourcen verschwenden, indem man Planungen erstellt, die man später wieder ändert. Im Allgemeinen muss der Teamleiter zum Team kommen und die nötigen Schätzungen einholen, die er für die Planung braucht. Die Schätzung wird immer von derjenigen Person gemacht, die später auch die Implementierung vornehmen wird. Ferner werden stets die Features mit der höchsten Priorität zuerst geplant und bearbeitet.

XP-Entwicklungsstrategie

Die Entwicklungsstrategie beruht auf 3 XP Praktiken. Kontinuierliche Integration, Collective Code Ownership und Pair Programming.

Durch die kontinuierliche Integration wird sichergestellt, dass nach jedem Entwicklungstag ein lauffähiges System zur Verfügung steht, auf welchem 100% aller Tests fehlerfrei ablaufen. Collective Code Ownership trägt dazu bei, dass alle Entwickler ermutigt werden, den bestehenden Code zu verändern, wenn ihre momentane Aufgabe dadurch vereinfacht wird. Es gibt kein Stück Code, das unantastbar ist. Pair Programming sorgt schlussendlich dafür, dass Kommunikation entsteht. Der Code ist das Ergebnis der Diskussion zwischen den Programmierern. Gleichzeitig wird damit auch die Fehlerrate gesenkt und Sackgassen vermieden. Ferner lernen die Programmierer von ihren Partnern.

XP-Design Strategie

Grundsätzlich sollten wir darauf zielen, das einfachste Design, das irgendwie möglich ist, zu finden. Grundsätzlich designen wir für heute und nicht für morgen. Morgen wissen wir genauer wo wir stehen und können das Design dann noch erweitern. Dies führt zu der folgenden Strategie:

1. Schreibe einen Test, damit Du weißt, wann du fertig bist. Einen Teil des Designs müssen wir natürlich schon bestimmen, um den Test zu schreiben.
2. Designe und implementiere nur so viel, bis der Test läuft.
3. Wiederhole 1 und 2.
4. Falls Du eine Möglichkeit siehst, das Design zu vereinfachen, mach es.

Die klassischen Software Engineeringtechniken zielen darauf ab, dass man das Programm später im Projekt so wenig wie möglich verändern muss. Bei XP ist das genau umgekehrt. XP geht davon aus, dass man das System täglich verändert (embrace change). Kent Beck: *„A day without refactoring is like a day without sunshine."* Der Unterschied ist, das XP so funktioniert, dass die Kosten für ein Refactoring sehr gering sind.

XP-Teststrategie

In XP hat das Testen eine etwas andere Funktion als in klassischen Ansätzen. Die Entwickler und Benutzer schreiben Tests, um Programme zum Laufen zu bringen und nicht, um Programme zu testen. Der Schwerpunkt liegt auf dem Verhalten des Systems, das durch Tests überwacht wird, und nicht auf den Tests an sich. Im Allgemeinen gehen Programmierer davon aus, dass ihr Code korrekt funktioniert und denken, dass es sich nicht lohnt, Tests zu schreiben. Besonders da dies eine relativ mühsame Arbeit sein kann. In XP sind die Tests die Grundlage des ganzen Projekts. Sobald die Tests gemacht sind, hält sich die Arbeit zum Testen in Grenzen, da alle Tests automatisch ablaufen. Die Tests sind enorm wichtig, da sie vorgeben welche Arbeit zu tun ist. Man bearbeitet den Code so lange bis alle Tests ohne Fehler funktionieren. Durch die automatischen Tests bekommt man sofort Feedback und entwickelt ein gestärktes Vertrauen in den Code. Oft wird im XP Team ein Tester bestimmt, der den Kundenvertretern hilft, ihre Verhaltenstests zu implementieren. Auch beim Testen tritt zum Vorschein, dass ein wichtiger Aspekt von XP die Weiterentwicklung der einzelnen Entwickler ist. So werden sie im Laufe des Projektes lernen, was sie zu testen haben und wie sie am besten ihre Tests entwickeln. Die Modultests und die Funktionstests bilden die Grundlage des Testens in XP. Je nach Projekt kommen noch Stresstests oder Überlastungstests dazu.

3. Ausblick

Zusammenfassend kann man sagen, dass XP entwickelt wurde, um die Entwicklung von Software billiger, schneller und flexibler zu gestalten und das Risiko zu senken. Die Grundlage bietet der Ansatz, dass man die Kostenkurve des Entwicklungsprojektes durch verschiedene Praktiken verändern kann. Die Tatsache, dass XP Funktioniert, ist dank zahlreichen Projekten erwiesen.

Basierend auf den Werten Kommunikation, Einfachheit, Feedback und Mut, schafft es XP, die Ideen des klassischen Software Engineerings zu hinterfragen und die Methodik radikal zu ändern. Folgende Statements wären in der klassischen Entwicklung undenkbar:

„Die Spezifikation der Software wird nur oberflächlich gemacht und im Verlaufe des Projektes angepasst."

„Programmiere kurzsichtig, implementiere nur was du heute brauchst, denke nicht an Morgen"

„Alle Entwickler dürfen den bestehenden Code jederzeit ändern, wenn dies zur Einfachheit beiträgt."

Neben der Tatsache, dass XP funktioniert kommt noch dazu, dass es viel Spass machen muss nach XP zu arbeiten. Das Team ist die Grundlage des Projektes. Wenn das Team nicht zusammen arbeiten kann, funktioniert XP nicht.

Wir werden sehen, was die Zukunft bringt. Ich bin jedenfalls überzeugt, dass wir noch viel über XP hören werden.

Literaturverzeichnis

Extreme Programming Explained – EMBRACE CHANGE

Kent Beck, 2000. Addison-Wesley, ISBN 201-61641-6

www.extremeprogramming.org

XP: A Gentle Introduction

www.xprogramming.com

An Extreme Programming Resource