# XP und RUP – Passt das zusammen?

Markus Barchfeld
Roland Sand
Johannes Link

Andrena Objects GmbH, Albert-Nestler-Straße 9, 76131 Karlsruhe
{markus.barchfeld|roland.sand|johannes.link}@andrena.de

**Zusammenfassung.** Planbare Softwareentwicklung ist ohne ein Vorgehensmodell nicht möglich. Rational Software entwickelte mit dem Rational Unified Process (RUP) ein Prozessmodell, das viele Entwicklungstätigkeiten, Ergebnisse und Abhängigkeiten im Detail beschreibt. Eine andere Strömung, die einen minimalen Prozess für ausreichend hält, wird durch eXtreme Programming (XP) immer populärer. Dieser Artikel beleuchtet Unterschiede und Gemeinsamkeiten und versucht, Kriterien für die Anwendung des einen oder des anderen Vorgehensmodells herauszustellen.

## 1 Einführung

Seit etwas mehr als einem Jahr sorgt ein neues Schlagwort für Furore unter den Softwareentwicklern: *Extreme Programming* (XP) [1]. XP ist in vielerlei Hinsicht „extrem“: Auf der einen Seite werden bestimmte Praktiken der Softwareentwicklung weiter getrieben als anderswo (siehe 2.1), zum anderen werden einige etablierte Verfahrensweisen der Softwaretechnik vollständig fallengelassen. Diese Radikalität bringt XP häufig den Vorwurf ein, es mache die konzeptlose Hackerei zum Prinzip.

Ein schwergewichtigerer Softwareentwicklungsprozess ist der *Rational Unified Process* (RUP) [5]. Mit dem RUP versucht die Firma Rational-Software seit ca. drei Jahren ein detailliertes Vorgehensmodell als Standard zu etablieren und durch den Verkauf eines dazugehörigen Produktes zu unterstützen.

Während viele Entwickler mit der schlanken und entwicklungsorientierten Vorgehensweise von XP sympathisieren, muss die Managementebene noch von dessen Adäquatheit überzeugt werden. Zudem zeigt die lebhafte Diskussion in News-Gruppen und Mailing-Listen, dass XP keineswegs ein *Silver Bullet* [2] darstellt und dass es häufig Situationen und Projekte gibt, in denen die „reine“ XP-Lehre nicht adäquat erscheint. Dieser Artikel ist ein Versuch, den gegenwärtigen Stand der Diskussion für und wider XP im Vergleich zu RUP zusammenzufassen, und zu beleuchten, ob eine Verbindung der beiden machbar und überhaupt wünschenswert ist.

# 2 Grundlagen

## 2.1 XP

XP ist eine Kombination aus bekannten Techniken zu einem neuartigen Softwareprozess [1]. Die Entwicklung eines Lohnabrechnungssystems bei DaimlerChrysler diente als „Pilotprojekt“. Spätestens seit Kent Beck‘s Keynote „eXtreme Programming: Everything You Were Told About Software Engineering Was Wrong” auf der OOP’99 in München ist das Thema auch in Deutschland in aller Munde.

Einen sehr hohen Stellenwert hat bei XP die direkte Kommunikation zwischen allen Beteiligten, sowohl zwischen Entwicklern untereinander als auch zwischen Entwicklungsteam, Kunden und Anwendern. Diese wird u. a. durch kleine Teams, paarweises Programmieren, gemeinsames Code-Eigentum und der unittelbaren Mitwirkung eines Kunden oder späteren Anwenders gefördert. Kommunikation soll bei XP den größten Teil der Dokumentation überflüssig machen.

XP legt wert auf ein schlankes System. Nach dem Motto „*Do the simplest thing that could possibly work*“ ist eine einfache Lösung mit Entwicklungspotential heute besser, als von Beginn an etwas allgemeines zu entwickeln, das morgen nicht benötigt wird - „*You ain’t gonna need it*“. Angenommen wird hierbei, dass es möglich ist, die klasischerweise angenommene exponetielle Kostkurve[1] durch das Zusammenspiel aller XP-Techniken auf eine stark abgeflachte Kurve zu reduzieren. Ob dies wirklich möglich ist, ist einer der umstrittensten Punkte.

Auch spielt bei XP eine direkte und möglichst schnelle Rückkopplung für alle Aktivitäten eine zentrale rolle. Der Programmierer kann mit Hilfe von Tests die Qualität seiner Implementierung unmittelbar erkennen; der Kunde bekommt durch ehrliche Aufwandsschätzungen der Entwickler Informationen über die Kosten seiner Anforderungen. Durch eine möglichst frühe Auslieferung kann der Anwender sehr bald Fehlentwicklungen aufdecken.

Selbstvertrauen ist eine wichtige Eigenschaft, die jeder XP-Entwickler haben sollte. Es ist beispielsweise notwendig, konsequent einen Programmteil zu löschen und durch einen neuen zu ersetzen, wenn dieser das Gewünschte zu umständlich tut. Es erfordert oft auch Mut, neue Ideen einzubringen und zu experimentieren, wenn dadurch das ganze System beeinflußt wird.

Aus diesen Grundprinzipien werden die Techniken des XP abgeleitet, zu den wichtigsten gehören:

---

[1] Gemeint ist der Zusammenhang zwischen Kosten einer Änderung und wann (Analyse, Design, Programmierung, Test) diese Änderung im System vorgenommen wird.

- Das System wird stufenweise in kurzen Iterationen erweitert. Vor jedem Zyklus stehen Anforderungen des Kunden, von denen die Entwickler den Arbeitsaufwand schätzen. Der Kunde wählt dann die tatsächlich anzugehenden Aufgaben aus. Das Ergebnis jeder Runde ist ein ablauffähiges Softwaresystem.
- Der Entwickler schreibt vor dem Hinzufügen einer neuen Funktionalität einen Test, welcher das gewünschte Verhalten festlegt. Mit Hilfe von Werkzeugen kann auf Knopfdruck jederzeit kontrolliert werden, ob der Test erfolgreich abläuft. Die Arbeit ist erst dann erledigt, wenn der Test positiv ist. Nach jeder abgeschlossenen Änderung und vor jeder Auslieferung müssen alle bisherigen Tests erfolgreich laufen.
- Refaktorierung (*Refactoring*), d.h. die Umstrukturierung eines Programm ohne seine Funktionalität zu verändern, ist wesentlicher Bestandteil der täglichen Arbeit. Durch meist kleine Schritte, wie Variablen umbenennen, Felder kapseln, Duplikate extrahieren oder Vererbungshierarchien durch Delegation ersetzen, wird der Programmcode kontinuierlich klarer und lesbarer, d. h. der Code dokumentiert sich zunehmend selbst. Die Tests sorgen nach jedem Schritt dafür, dass sich keine Fehler einschleichen.
- Jedes Stück Software wird paarweise entwickelt. Einer bedient die Tastatur und die Maus und überlegt sich die richtige Ausformulierung einer Methode. Der andere denkt eher strategisch, z. B. welche Fälle mit einem Test überprüft werden können, oder welche Möglichkeiten es zur Vereinfachung gibt.

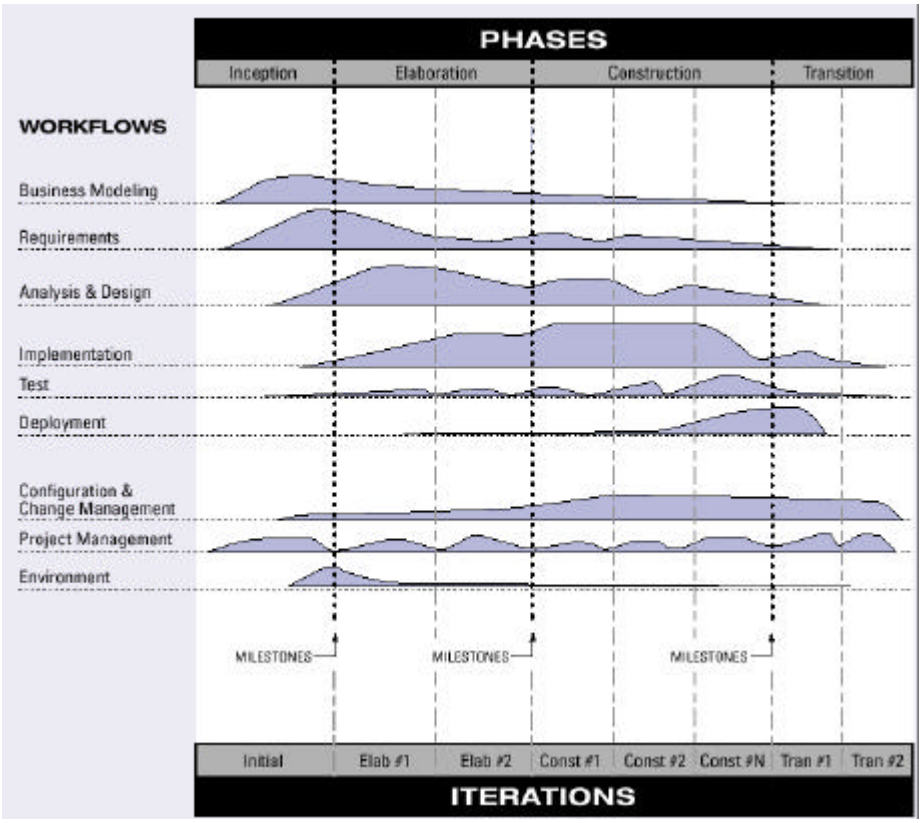
## 2.2 RUP

Der RUP [5], als ein Prozessmodell zur Softwareentwicklung, beschreibt Workflows, die mittels UML-Aktivitätendiagrammen[2] definiert sind. Es gibt in ihm sechs Kernworkflows die das Vorgehen für Geschäftsmodellanalyse, Anforderungsbestimmung, Analyse und Design, Implementierung, Test und Installation (Deployment) festlegen. Elemente der Workflows sind Rollen (Workers), Aktivitäten und Artefakte.

Der RUP legt bestimmte Rollen, Aktivitäten und Artefakte des Entwicklungsprozesses fest. Einer Rolle sind bestimmte Tätigkeiten und Artefakte zugewiesen. Eine Aktivität ist eine Arbeitseinheit, die von einem Beteiligten in einer bestimmten Rolle ausgeführt wird. Das Ergebnis ist ein Artefakt. Artefakte können Dokumente (z.b. Risikoliste), Modelle (z.B. Anwendungsfallmodell) oder Modellelemente (Klasse aus dem Klassenmodell) sein. Für manche der Artefakte liefert der RUP Vorlagen, z.B. um die Ergebnisse eines Code Reviews festzuhalten.

---

[2] In älteren Versionen des RUP werden noch informelle Grafiken zur Workflowbeschreibung verwendet.




Das obige Diagramm (aus [7]) beschreibt die Aufwandsanteile, die einzelne Workflows über den Verlauf der Entwicklung haben sollten. Die Entwicklung ist in vier Phasen eingeteilt: Anfangsphase (*inception*), Ausarbeitungsphase (*elaboration*), Konstruktion (*construction*) und Übergabe (*transition*). Jede Phase besteht aus mindestens einer Iteration, wobei eine Iteration ähnlich dem Wasserfallmodell aus den Workflows zusammengesetzt ist.

Ausgangspunkt und Grundlage für die Entwicklung sind Anwendungsfälle. Sie werden während der Entwicklungszeit weitergepflegt. Der RUP kann deshalb als anwendungsfallgetrieben bezeichnet werden.

Weiterhin beschreibt er sich als architekturzentriert. Die Architektur gliedert das System in Komponenten und Subsystem und fügt es in die Systemlandschaft ein. Die Darstellung der Architektur soll das Verständnis der Schnittstellen und Wiederverwendung ermöglichen. Wo möglich sollen Komponenten verwendet werden.

Der RUP kann konfiguriert und auf die Bedürfnisse einer Firma oder Projektes zugeschnitten werden. Die angegebenen Workflows können, wenn sie nicht benötigt

werden, weggelassen oder verkürzt werden. Die Veränderung bestimmter Workflows wird nur für Sonderfälle empfohlen.

## 3 Vergleich von XP und RUP

Bevor mit einem Vergleich begonnen werden kann, müssen wir zunächst festlegen, was wir vergleichen. Bei XP gehen wir davon aus, daß er kompromisslos, wie in [1] beschrieben, durchgeführt wird. Bei RUP gehen wir von einem typischen RUP aus, bei dem die meisten der in [7] beschriebenen Elemente angewandt werden.

Die auffälligste Gemeinsamkeit beider Modelle ist das inkrementelle, iterative und anwendungsfall-getriebene Vorgehen. Während jedoch im RUP versucht wird, den Großteil der Anforderungen möglichst früh festzuschreiben, hat XP die Devise, dem Kunde zu jeder Zeit einen völligen Richtungswechsel zu erlauben.

Architektur ist in beiden Modellen bedeutend. In der RUP Definition kommt das klar heraus, bei XP wird das erst auf den zweiten Blick deutlich: Zum einen versucht ein XP Projekt mit einem Architekturprototypen zu starten, zum anderen spielt hier die Systemmetapher eine ähnliche Rolle wie die Architekturbeschreibung beim RUP.

Hauptziel von XP ist ein minimaler Prozess, der die Entwickler möglichst wenig belastet. Der RUP hingegen erlaubt zwar ein Zuschneiden des Prozesses, fordert aber in jedem Falle eine strikte Einhaltung der Phasen und Kernworkflows.

Die Arbeitsergebnisse von XP sind der Code und die Tests. Designdokumente werden nur temporär erzeugt, zusätzliche Dokumente, z.B. für die Wartungsübergabe, haben geringe Umfang. Dagegen sind alle im RUP definierten Artefakte dauerhaft im Konfigurationsmanagement abgelegt. Visuellen Designmodellen unter Verwendung der UML kommt hierbei besondere Bedeutung zu.

In XP ist Spezialisierung unerwünscht, jedes Teammitglieder sollte alle Teile des Systems kennen. Wechselnde Programmiererpaare unterstreichen diesen Aspekt. Durch die Rollenverteilung im RUP wird Spezialisierung gefördert und zuweilen sogar verlangt.

Auch die Designstrategie ist unterschiedlich. XP verwirklicht evolutionäres Design, das einfach beginnt und auf die Änderungsfähigkeit des Programmcodes vertraut. RUP startet mit einem möglichst vollständigen Design (*„Big up front design“*). Designmodell und Implementierung werden dann laufend synchronisiert (Roundtrip).

XP stellt Kommunikation über Dokumentation. Auf diese Art werden auch Reviews durchgeführt: Durch Programmiererpaare und der Veränderbarkeit des Codes (gemeinsamer Codeeigentum) umfasst das Programmieren an sich schon den Review. Der RUP definiert dafür eigene Aktivitäten und Artefakte, die dem eigentlichen Entwickeln nachgeschaltet sind.

Management beschränkt sich in einem XP-Projekt auf wenige Aktivitäten. Diese umfassen hauptsächlich Kontrollen der Zeitschätzungen der Teammitglieder und das Anfertigen von Statistiken als Spiegel des Projektfortschritts. Im RUP kommt dagegen Detailplanung und Plankontrolle zum Einsatz.

In all diesen Aspekten zeigen sich die unterschiedlichen Schwerpunkte von RUP und XP: Im Zentrum von RUP steht die Bewältigung eines komplexen arbeitsteiligen, verteilten und lang andauernden Entwicklungsprozesses durch dessen Zergliederung in definierte Produktionsschritte mit definierten Ergebnissen. Dieser tayloristische Ansatz entspricht dem Bestreben des Managements nach Kontrolle und Vorhersagbarkeit.

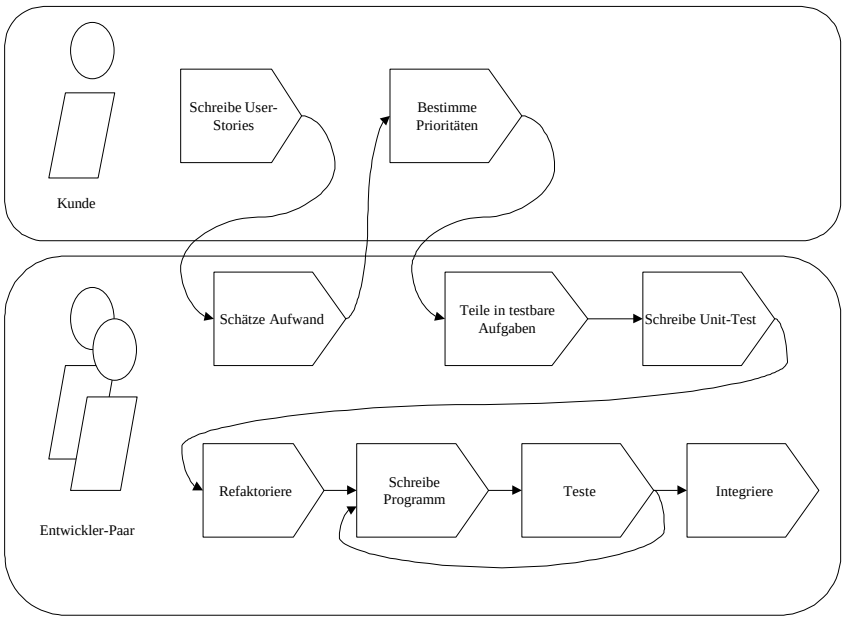
Im Zentrum von XP steht das Bestreben, die zentrale Tätigkeit des Programmierens im Team aufzuwerten, und die Entwickler mit Werkzeugen und Techniken auszustatten, die dieser zentralen Bedeutung gerecht wird.

## 4 Ist XP eine Instanz von RUP?

Der RUP ist ein (eingeschränkt) anpassbares Prozess-Framework, XP dagegen eine verbindliche Vorgehensbeschreibung. Deshalb stellt sich die Frage, ob man XP als einen RUP-Prozess definieren kann (siehe auch [6]).

Das RUP-Produkt besteht aus mehreren tausend HTML-Seiten. Es werden z. B. 27 Rollen definiert, die ca. 50 verschiedene Dokumentenarten hervorbringen können [2, Anhang A u. B]. Es kann als Ganzes verwendet werden, oder an eigene Bedürfnisse angepasst werden. Erlaubt sind u. a. Hinzufügen, Erweitern, Verändern und Wegnehmen von Arbeitsabläufen, Rollen, Dokumenten, Richtlinien und Bezeichnungen [2, S. 49 u. S. 225], solange Grundprinzipien wie z. B. iterative Entwicklung und Qualitätskontrolle nicht verletzt werden. Durch intensives Streichen insbesondere von Rollen und Dokumenten kann man sich in RUP an XP annähern.

Einen Arbeitsablauf einer XP-Iteration in der Konstruktionsphase beispielsweise, könnte man in RUP-typischer Weise so darstellen:




Die wichtigsten RUP-Prinzipien werden von XP erfüllt (vgl. Kap. 3). Daraus kann man schließen, dass XP tatsächlich eine Instanz von RUP ist. Tatsächlich deutet einges darauf hin [8], dass auch Rational erkannt hat, wie wichtig eine „Versöhnung“ von RUP und XP sein könnte. Für Unternehmen, die bisher in RUP investiert haben, könnte das ein Argument sein, XP nun ebenfalls als Option zu betrachten.

Die Philosophien, die hinter RUP bzw. XP stehen, unterscheiden sich jedoch deutlich voneinander. In der Praxis würde es sich daher kaum lohnen, aus RUP einen XP-Prozess durch Weglassen abzuleiten. Es empfiehlt sich stattdessen, je nach Rahmenbedingungen, sich entweder für RUP mit nur geringfügigen Anpassungen zu entscheiden oder den Regeln von XP zu folgen. Auch eine Mischform sollte man in Betracht ziehen. Entscheidungshilfen gibt das folgende Kapitel.

# 5 XP oder RUP oder XP und RUP?

Im vorangegangenen Kapitel wurde argumentiert, dass XP als RUP-Instanz betrachtet werden kann - oder auch nicht, je nach Perspektive. Für den pragmatischen Projektleiter ist diese eher theoretische Frage jedoch weniger interessant. Was ihn beschäftigt, ist die Frage, ob er das entwicklerfreundliche XP propagieren soll, oder ob eine formalerer Prozess, z.B. RUP, für das anstehende Projekt geeigneter ist. Im Folgenden beleuchten wir einige Entscheidungskriterien sowie die Möglichkeit, XP um Elemente des RUP zu ergänzen.

## 5.1 Ideale XP Rahmenbedingungen

Unter bestimmten Bedingungen scheint XP das ideale Vorgehensmodell zu sein, weil es sowohl Entwicklern als auch Kunden und Managern entgegen kommt. Diese Bedingungen sind u.a.:

- Mittlere Projektgröße (2 – 12 Entwickler)
- Kunde vor Ort
- Kommunikative Entwicklermentalität
- Neuentwicklung bzw. Ersetzen einer vorhandenen Applikation
- Objektorientierte Programmiersprache mit integrierter Umgebung zum Refaktorieren, Testen und Code Management

## 5.2 Probleme bei der Anwendung von XP

Sind diese Voraussetzungen nicht erfüllt, so kann der lehrbuchmäßige Einsatz von XP zu Problemen führen bzw. unmöglich gemacht werden. Einige Beispiele:

In einer an Hierarchien und starker Kontrolle orientierten Unternehmenskultur werden XP-Techniken wie *Pair Programming* und *Unit Testing* häufig nicht akzeptiert, da sie scheinbar den Aufwand für die Realisierung erhöhen. Auch kann man mit Entwicklern, die bestimmte XP-Prinzipien (z.B. gemeinsames Codeeigentum) ablehnen, kein XP betreiben.

Häufig ist die räumliche Verteilung der Entwickler so, dass die enge Kommunikation und Abstimmung stark behindert oder sogar verhindert wird.

Großprojekte bieten zahlreiche Problemstellen für XP:

- Ab einer gewissen Komplexität kann ein Entwickler nicht mehr alle wichtigen Systemdetails im Kopf behalten. Spezialisierungen sind unumgänglich.
- Die aufwendige Integration verschiedener Systemteile wird zum Bottleneck.
- Langlaufende Test-Suiten führen zu langen Feedback-Zeiten.

Auch ist bei bestimmten Projektarten eine exponentielle Kostenkurve kaum zu verhindern. Typische Beispiele sind kombinierte Hardware-/SW-Projekte, oder wenn ein System als Grundlage vieler anderer unabhängiger Projekte dienen muss (Framework-Entwicklung).

Gibt es keine Möglichkeit, den „Kunden vor Ort“ zu bekommen, so muss ein wesentlicher Bestandteil von XP, die Release- und Iterationsplanung, auf andere Weise durchgeführt werden.

## 5.3 Erweiterung von XP um RUP-Elemente

In einigen der oben genannten Problemfällen ist es denkbar, XP um Elemente des RUP zu erweitern.

### Großprojekte

Ein Großprojekt läßt sich häufig als Föderation aus Teilprojekten organisieren. Während die Gesamtarchitektur mit Hilfe von *UML Design Dokumenten*, *Change Control* und *Configuration Management* à la RUP konsistent gehalten wird, verwenden die Teilprojekte den XP-Prozess. Die zu erwartendende deutlich größere Anzahl von Fehlern werden mittels eines *Defect-Tracking*-Systems auf die einzelnen XP-Teams verteilt. Probleme bereiten dabei jedoch die Fragen:

- Wer ist der Kunde des Teilprojekts?
- Wie verfährt man bei globalen Refaktorierungen?
- Wie synchronisiert man die Iteration der einzelnen Teams?

### Räumliche Verteilung

Sitzen die Entwickler ständig oder zeitweise so weit auseinander, dass paarweises Programmieren nicht durchführbar ist, muss man auf andere Möglichkeiten zurückgreifen, um das 100%-Review-Prinzip von XP zu erfüllen. Herkömmliche Peer-Review-Verfahren können zeitweise eine Alternative darstellen.

### Kein Kunde vor Ort

Fehlt der Kunde vor Ort, so ist die Planungsphase im XP-Sinne nicht durchführbar. Erste positive Erfahrungen haben wir in diesem Falle mit folgendem Vorgehen gemacht: Die XP-Release-Planung wird durch eine Use-Case-Analyse ersetzt, der Kunde priorisiert die Use Cases, und die Iterationsplanung findet schließlich anhand der detaillierten Use Cases statt. Natürlich wächst dabei die Gefahr, dass man spekulativ entwickelt, weil der Kunde nicht immer für auftauchende Fragen zur Verfügung steht.

### Ist das noch XP?

Man sollte sich jedoch keine Illusionen machen: Ein Vorgehen, das XP so verändert wie in den vorangegangenen Abschnitten beschrieben, ist kein XP mehr. Das heißt, dass viele der positiven Auswirkungen von XP durch Veränderung leiden oder sogar völlig verschwinden. Erst die Erfahrungen der kommenden Jahre werden zeigen, wie weit XP verändert werden kann, ohne seinen eigentlichen Vorteil einzubüßen: Leichtgewichtig zu sein und gleichzeitig qualitativ hochwertige und kundenbestimmte Software hervorzubringen.

## 6 Fazit

Das gegenwärtige Forschungsgebiet Software Engineering liefert noch kein universelles Prozessmodell, das für alle Softwareprojekte gleichermaßen geeignet wäre. Manche Praktiken, wie z. B. die iterative und inkrementelle Entwicklung, haben sich aber als besonders nützlich erwiesen und werden von allen gängigen Vorgehensmodellen propagiert. Der Nutzen oder Unsinn vieler anderer Techniken muss dagegen von Fall zu Fall überdacht werden.

Die Diskussion über das, was sich unter dem neuen Schlagwort "XP" verbirgt, gibt den Entwicklern hilfreiche Anregungen, ihre individuelle Arbeitsweise zu verbessern und den Unternehmen Ideen, Projekte effektiver zu organisieren.

Die aktuellen Bemühungen von Rational, den RUP XP-kompatibel zu machen [8] und die Diskussionen im XP-Umfeld über Erweiterungen der XP-Praktiken, zeigen, dass sich die beiden Sichtweisen nicht widersprechen, sondern ergänzen. Variationen und Experimente sind aus unserer Sicht nicht nur erlaubt, sondern notwendig und erwünscht.

## Literatur


**[1]** Kent Beck: *Extreme Programming Explained: Embrace Change*. Addison Wesley, 1999.

**[2]** F.P. Brooks: *No silver bullet*. IEEE Computer 20 (April 1987), pp. 23 – 29.

**[3]** N.M. Josuttis: *Extreme Programming: Ein Gespräch mit Kent Beck*. OBJEKTspektrum, Juli/August 1999.

**[4]** Martin Fowler: *Refactoring – Improving the Design of Existing Code*. Addison Wesley, 1999.

**[5]** Philippe Kruchten: *The Rational Unified Process, An Introduction*. Addison Wesley. Addison Wesley, 2000.

**[6]** Robert Martin: *XP: A Lightweight Instance of RUP*. C++ Report, June 2000.

**[7]** Rational Software Corp.: http://www.rational.com/products/rup

**[8]** Martin Fowler: http://www.martinfowler.com/articles/xp2000.html