Extreme Programming – Back to Basics?

Bernhard Rumpe

Software & Systems Engineering, Informatik
Technische Universität München, www.rumpe.de

Zusammenfassung: Die Suche nach einer optimalen Vorgehensweise bei der Softwareentwicklung bringt öfter neue Vorschläge für Vorgehensmodelle hervor. Extreme Programming ist eine radikale, aus der Praxis motivierte Variante, die versucht „best practices“ der Software-Entwicklung in möglichst schlanker Weise zu kombinieren und soweit wie möglich auf organisatorischen Ballast zu verzichten. Aufgrund der hohen Praxisrelevanz von XP scheint eine wissenschaftliche und kritische Aufarbeitung der Thematik geboten. Dieser Beitrag exploriert einige der Themenfelder von XP und stellt somit einen Baustein dieser Aufarbeitung dar.

1 Woher kommt und was ist Extreme Programming?

Extreme Programming (in Kurzform: XP) ist eine sogenannte „leichgewichtige“ Softwareentwicklungs-Methode. Ihre Entwicklung und Verbreitung wird zumindest in der Anfangsphase von XP den drei Softwareentwicklungs-Gurus Kent Beck, Ron Jeffries und Ward Cunnigham zugeschrieben. Obwohl XP als Vorgehensweise u.a. bei Softwareprojekten einer Schweizer Bank definiert und verfeinert wurde, zeigt schon die Namensgebung eine starke Beeinflussung durch die nordamerikanische Softwareentwicklungs-Kultur. Denn trotz des gewählten Namens ist XP keineswegs nur eine Hacker-Technik, sondern besitzt einige sehr detailliert ausgearbeitete methodische Aspekte. Diese erlauben es ihren Befürwortern zu postulieren, dass durch XP mit verhältnismäßig wenig Aufwand qualitativ hochwertige Software unter Einhaltung der Budgets zur Zufriedenstellung der Kunden erstellt werden kann.

XP erfreut sich gerade in der Praxis einer stetig steigenden Popularität. Zahlreiche Bücher sind erschienen bzw. in Vorbereitung. Darunter sind [Be99, JAH00, BF00] die teilweise unterschiedliche Aspekte der Thematik sowie jeweils aktuelle Wissensstände der sich noch stark in Weiterentwicklung befindlichen Methodik detailliert betrachten. Die Popularität von XP in der Praxis zeigt sich auch anhand zahlreicher Artikel in Computerzeitschriften wie [We99, Ec00, EH00, EH01], die einen oft lesenswerten Überblick über den jeweils aktuellen Stand von XP oder einen einführenden Vergleich mit alternativen Ansätzen beschreiben. Aktuelle Informationen lassen sich über mehrere im Internet eingerichtete Diskussionsgruppen und Informations-Sites zum Thema XP erfahren. Diese Informationen sind allerdings nicht notwendigerweise konsistent und nur wenig aufbereitet [We01, Cu01, Je01].

Obwohl derzeit nur wenige genaue Zahlen über XP-Projekte vorliegen, kann davon ausgegangen werden, dass XP heute vielen Softwareentwicklern, zumindest im nord-

amerikanischen und europäischen Raum, ein Begriff ist. Eine Reihe von Consulting-Firmen beginnen ihre Mitarbeiter und Kunden für zukünftige XP-Projekte vorzubereiten. Firmen mit Inhouse-Software-Entwicklung starten Pilot-Projekte, wobei dort aufgrund der teilweise radikal anderen Sichtweise von XP gegenüber klassischen Vorgehensweisen eine deutliche Verunsicherung über die Implikationen besteht. Die Techniken und Konzepte von XP führen zu einer relativ starken Polarisierung der Meinungsbildung. Auf der einen Seite glauben Softwareentwickler, XP erhebe ihre bisherige, informelle Vorgehensweise (Hacking) zur Ingenieurs-Disziplin. Auf der anderen Seite wird XP verdammt, weil es alles über Bord werfe, was in den letzten Jahrzehnten an Vorgehensmethodiken so mühsam erarbeitet worden ist.

Richtig ist, dass XP eine leichtgewichtige Softwaremethode ist, die explizit als Gegengewicht zu schwergewichtigen Methoden, wie dem Rational Unified Process [Kr00] oder dem V-Modell [Ve00] positioniert ist.

In Abschnitt 2 wird ein kurzer Überblick über die wesentlichen Elemente von XP gegeben. Im Abschnitt 3 werden zwei interessante technische Aspekte, Testen und Refactoring, genauer beleuchtet. Der Abschnitt 4 behandelt die Frage einer wissenschaftlichen Aufarbeitung von XP und des Umgangs mit XP in der Lehre. Im fünften Abschnitt werden einige Ansätze aufgezeigt, die XP mit traditionellen Techniken verbinden, und so einen Ausblick auf Verbesserungsmöglichkeiten von XP gegeben.

2 XP- Ein Überblick

XP ist nicht einfach eine Neuauflage des Hacking, sondern eine leichgewichtige Softwareentwicklungs-Methode. Sie verzichtet auf eine Reihe von Elementen der klassischen Softwareentwicklung, um so eine schnellere und effizientere Kodierung zu erlauben. Die dabei entstehenden Defizite versucht sie durch stärkere Gewichtung anderer Konzepte (insbesondere der Testverfahren) zu kompensieren. Auch das Leichgewicht XP besteht aus einer Reihe von Einzelkonzepten, die alle vorzustellen den Rahmen dieses Überblicks sprengen würde. Stattdessen ist auf entsprechende Literatur (siehe eine Auswahl im Literaturverzeichnis) verwiesen.

XP versucht explizit nicht auf neue und damit nicht ausreichend erprobte methodische Konzepte zu setzen, sondern integriert weitgehend bewährte Techniken zu einem Vorgehensmodell, das auf Wesentliches fokussiert und auf organisatorischen Ballast soweit wie möglich verzichtet. Weil der Programmcode das ultimative Ziel einer Softwareentwicklung ist, fokussiert XP von Anfang an genau auf diesen Code. Im Gegensatz dazu wird Dokumentation als aufwendiger Ballast betrachtet. Eine Dokumentation ist aufwendig zu erstellen und oft sehr viel fehlerhafter als der Code, weil sie nicht automatisiert analysier- oder testbar ist. Zusätzlich verhindert sie eine flexible Weiterentwicklung des Systems eine schnelle Reaktion auf neue oder veränderte Anforderungen der Kunden, wie es in der Praxis häufig der Fall ist. Folgerichtig wird in XP-Projekten keine Dokumentation erstellt. Im Ausgleich dafür wird Wert auf eine gute Kommentierung des Quellcodes durch Coding Standards und eine umfangreiche Test-Sammlung gelegt.

Die primären Ziele von XP sind altbekannt: Effiziente Entwicklung qualitativ hochwertiger Software und unter Einhaltung von Zeit- und Kostenbudgets. Welche Mittel dazu eingesetzt werden, wird anhand der *Werte*, der *Prinzipien* und der *grundlegenden Aktivitäten* kurz vorgestellt. Die vier Werte sind:

- *Kommunikation*: Permanente und intensive Kommunikation der Entwickler untereinander, sowie mit den Kunden erlaubt schnellstmögliches Feedback sicherzustellen, unnötige Funktionalität zu verhindern, entstehende Probleme so schnell wie möglich zu lösen und das Problem der fehlenden Dokumentation zu mildern.
- *Einfachheit*: Die Software soll so einfach wie möglich gestaltet werden, keine Vorbereitung möglicher zukünftiger Erweiterungen, keine redundante oder unnötige Funktionalität und keine redundanten Strukturen sind geduldet. Dadurch bleibt das System einfach und wartbar. Dies basiert auf der XP-Annahme, dass es effizienter ist, heute etwas einfaches zu erstellen und morgen etwas mehr Aufwand zu investieren, um Änderungen einzubauen, als heute Komplexes zu entwickeln, das morgen nicht oder nicht in der antizipierten Form genutzt wird.
- *Feedback*: Viele heutige Projekte scheitern an Missverständnissen zwischen Entwickler und Anwendern. Evolutionäre Entwicklung des Systems in möglichst kleinen Releases und eine permanente Verfügbarkeit der Kunden erlaubt schnelles Feedback und dadurch flexible Steuerung des Projektfortschritts. Eine weitere wichtige Quelle des Feedbacks ist die Entwicklung von Tests auf verschiedenen Ebenen (Unit-Tests, Test-Stories), um zu prüfen, ob die realisierte Funktionalität korrekt und robust ist und gegebene Anforderungen erfüllt.
- *Eigenverantwortung*: Die Entwickler sind angehalten, eigenverantwortlich zu handeln. Das impliziert, in Absprache mit dem Kunden Funktionalitäten anzupassen, Prioritäten zu aktualisieren und Pläne zu überdenken. Verantwortungsbewusstsein beinhaltet auch, dass jeder Programmierer den Überblick über das Gesamtsystem hat und sich zutraut, von Kollegen entwickelten Code zu modifizieren.

XP kennt drei wesentliche Rollen: den Projektleiter, den Kunden und den Entwickler. Der *Projektleiter* ist verantwortlich für das Management und die Koordination des Projekts, er verwaltet Ressourcen, Kosten und Zeitpläne, ist aber im Normalfall ebenfalls als Entwickler tätig. Wenigstens ein *Kunde* ist permanent ansprechbar, um schnell aufkommende Fragen zu klären. Idealerweise sitzt er mit den Entwicklern im selben Raum und entwirft funktionale Tests für die Software (User-Stories). Die *Entwickler* tragen die Hauptlast des Projekts. Sie kodieren, testen, entwerfen und hören dem Kunden aufmerksam zu. Durch die Besonderheiten von XP gibt es eine Reihe von notwendigen Rahmenbedingungen für XP-Projekte. So scheint nur eine Maximalzahl von 10 Programmierern in einem Projekt sinnvoll, obwohl bereits größere XP-Projekte erfolgreich abgeschlossen wurden. Es muss ein Kunde zur intensiven Einbindung in das Projekt zur Verfügung stehen. Die Räumlichkeiten erlauben intensive Kommunikation und Kontakte und der Kunde verzichtet auf eine ausführliche Dokumentation von Analyse und Entwurf.

XP in Reinkultur beschreibt *vier wesentliche Aktivitäten*:

- *Kodierung*: Das System wird in kleinen Schritten inkrementell erweitert. Kodierungsstandards und regelmäßige Durchläufe aller Tests, die gemeinsam mit dem Code entwickelt werden, sind wesentlicher Bestandteil der Kodierung. Modifikationen am Code werden mit Hilfe von Refactoring-Techniken [Fo99] durchgeführt.
- *Testen*: Jedes Programmelement besitzt automatisierte Tests. Komponententests entstehen gemeinsam mit dem Code. Kunden schreiben funktionale Tests, die die Geschäftslogik prüfen. Tests sind in XP methodisch besser verankert und integriert, als dies bei anderen Vorgehensweisen der Fall ist.

- *Zuhören*: Kommunikation zwischen den Entwicklern, sowie mit dem Kunden ist essentiell.
- *Design*: Ist Teil der täglichen Programmiertätigkeit. Während der Design-Aktivität wird u.a. die Systemlogik organisiert. Ein explizites Modell oder Design-Dokument wird aber nicht erstellt.

Folgende *fundamentale Prinzipien* werden propagiert:

- *Schnelles Feedback*: Erlaubt eine kontinuierliche Projektsteuerung, u.a. durch Priorisierung der Anforderungen.
- *Einfachheit*: Fördert die Klarheit und Eleganz des Codes.
- *Inkrementelle Änderungen*: Verhindern die Problemstellungen eines Big-Bang und erlauben einen messbaren Fortschritt.
- *Änderbarkeit unterstützen*: Um damit Flexibilität zu erhöhen.
- *Qualitativ hochwertige Ergebnisse*: Wird angestrebt durch eine Reihe von geeigneten Maßnahmen.

Auf Basis des vorgeschlagenen Wertesystems, der Rollenverteilung und der groben Aktivitätsbeschreibung definiert XP eine Reihe von *Entwicklungspraktiken*, von denen ein Teil hier genannt wird:

- Das *Planspiel*: Dient zur Erhebung von Anforderungen. Kunden entscheiden über die zu realisierende Funktionalität, ihre Priorisierung und die daraus erwachsende Release-Planung. Entwickler schätzen und entscheiden über die Aufwände, Konsequenzen und Vorgehensweise zur Implementierung der gewünschten Funktionalitäten.
- *Metaphern*: Sind dazu gedacht Grundprinzipien im Code durch anschauliche Begrifflichkeiten zugänglich zu machen.
- *Pair-Programming*: Je zwei Personen sitzen an einem Rechner: Abwechselnd tippt der eine, während der andere ihm über die Schulter schaut. Entwürfe werden so schneller und besser durchdacht, Fehler leichter erkannt, Neulinge können von Kennern effizient in den Code eingearbeitet werden, und die Kommunikation ist intensiv.
- *Testen, testen, testen*: Wird im nächsten Abschnitt genauer behandelt.
- *Refactoring*: Siehe ebenfalls nächster Abschnitt.
- *Gemeinsamer Codebesitz*: Ist eine logische Konsequenz des Pair-Programming und erlaubt eine schnellere Anpassung der Software bei geänderten Anforderungen.
- *Kleine Releases* (idealerweise im Wochentakt): Erlauben schnelles Feedback mit dem Kunden.
- *Kontinuierliche Integration*: Sichert, dass Systemteile sich nicht inkompatibel auseinander entwickeln.
- *Max. 40 Stunden Woche*: Sichert die Motivation der Entwickler und optimiert so die Leistungsfähigkeit der Entwickler und die Qualität der Ergebnisse.
- *Ständig verfügbarer Kunde*: Notwendig für ein intensives Feedback und zum Ausgleich der fehlenden Dokumentation.
- *Kodierungsstandards*: Sind eine notwendige Vorbedingung für gemeinsamen Codebesitz und den Ersatz von Dokumentation durch den Code.

Nach dieser Übersicht über wesentliche Elemente von XP werden im nächsten Abschnitt zwei interessante, bislang nicht diskutierte technische Aspekte vorgestellt.

3 Testen und Refactoring

Unter den in XP genutzten Konzepten spielen gerade die Entwicklung automatisierter Tests und die Verbesserung der Software durch Refactoring eine wesentliche Rolle. Beide Konzepte greifen intensiv ineinander, weil die Prüfung der Korrektheit von Refactoring-Maßnahmen ohne eine existente Test-Sammlung nicht praktikabel ist.

Testen in XP

Testen ist eine der wichtigsten Tätigkeiten in XP-Entwicklungsprojekten. Ziel der Testaktivität ist die Erstellung von automatisierten Tests mit einer möglichst vollständigen Überdeckung der Funktionalität des entwickelten Programms. Tests sind auf mehreren Ebenen zu entwickeln. Beginnend mit einfachen Methoden-Tests über Unit-Tests bis hin zur Überprüfung der Korrektheit von Geschäftsabläufen durch auf User-Stories basierenden Tests muss alles vertreten sein.

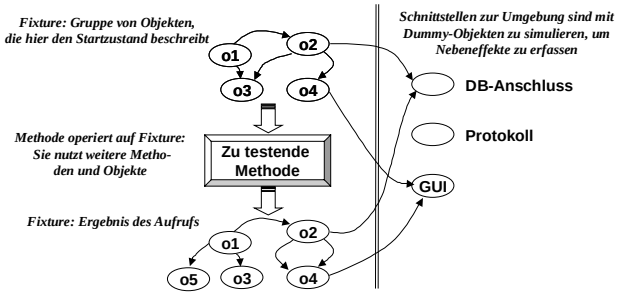

Abb. 1: Typische Form eines automatisierten Tests

In XP sind alle Tests automatisiert. Damit wird es möglich, jederzeit und damit auch nach kleinen Änderungen die Korrektheit des Systems durch Tests zu prüfen. Tests und Code werden parallel entwickelt, um so eine Testüberdeckung besser zu garantieren. Black-Box-Tests werden typischerweise vor der Methoden-Implementierung, White-Box-Test parallel oder nach der Methoden-Implementierung realisiert. Der Kunde beschreibt User-Stories, die im Idealfall von Kunden selbst dazu genutzt werden, Test-Cases für Geschäftsanwendungen zu realisieren. Für effizientes Management und zur Definition von Tests stehen für verschiedene Sprachen Test-Frameworks zur Verfügung. JUnit für Java gilt heute als am bekanntesten [Junit01]. Auch bei der Verwendung von Test-Werkzeugen gilt in XP das Prinzip der Einfachheit. Dies impliziert den Verzicht auf komplexe Tools und die Benutzung des einfachen Frameworks JUnit, das in der Entwicklungssprache selbst geschrieben wurde.

Bieten Tests eine gewisse Überdeckung der Systemfunktionalität, so kann die Gesamtheit der Tests als ein Modell des Softwaresystems verstanden werden. Anders als explizite Modelle, die z.B. in UML repräsentiert werden, ist ein Test-Modell sehr implizit, hat aber den Vorteil der automatisierten und wiederholbaren Prüfbarkeit. Dies ist insbesondere bei der Weiterentwicklung des Systems von Vorteil. Der Verzicht auf die Zuordnung von Code zu Verantwortlichen (Besitzern) ist nur möglich, weil korrekte Testläufe einem Entwickler eine gewisse Sicherheit geben, dass er fremden Code nicht in unzulässiger Weise verwendet hat. Insbesondere bei der Verbesserung der Struktur eines Systems durch Refactoring-Techniken ist eine gute Sammlung von Tests zwingende Voraussetzung. Die Vorteile einer guten Test-Sammlung zeigen sich erst später im Projekt. Es erfordert aber einigen initialen Aufwand eine Testumgebung einzurichten (siehe Abb.

1), und daher einiges an Disziplin, auf interaktives Debugging z.B. durch Test-Ausgaben zu verzichten und stattdessen automatisierte Tests zu schreiben.

Refactoring

Der Wunsch nach Techniken zur inkrementellen Verbesserung und Modifikation von Programmcode ist fast so alt wie die Programmierung selbst [Ba85]. Ziel einer transformationellen Software-Entwicklung ist die Zerlegung des Software-Entwicklungs-Prozesses in kleine, systematisch durchführbare Schritte, die aufgrund lokaler Anwendung überschaubar werden. Das empfehlenswerte Buch [Fo99] beschreibt Transformationstechniken auf Java-Basis. Es erlaubt die Migration von Code entlang von Klassenhierarchien, die Zusammenlegung oder Teilung von Klassen, die Verschiebung von Attributen, das Expandieren oder Falten von Code-Teilen in eigene Methoden, und ähnliches mehr (siehe Beispiel in Abb. 2). Die Stärke der Refactoring-Techniken entsteht durch die Überschaubarkeit der einzelnen Schritte und durch die flexible Kombinierbarkeit zu großen, zielorientierten Verbesserungen der Software-Architektur.

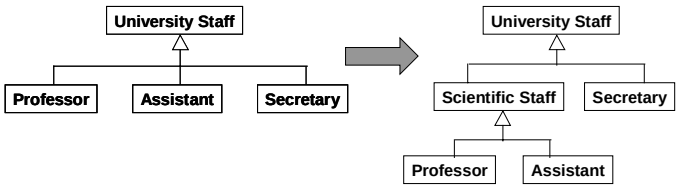

Abb. 2: Restrukturierung einer Klassenhierarchie als korrektheitserhaltendes Refactoring

Das Ziel des Refactoring-Konzepts sind semantikerhaltende Transformationen eines bereits existenten Programms. Refactoring dient nicht zur Erweiterung der Funktionalität, sondern zur Verbesserung der Qualität des Entwurfs unter Beibehaltung der Funktionalität. Es ergänzt damit die normale Programmiertätigkeit (siehe Abb. 3). Ziel ist die Implementierung der gewünschten Funktionalität unter Beibehaltung eines hohen Qualitätsstandards für Design und Architektur. Dadurch bleibt das System wartbar, einfach und seine Qualität hoch.

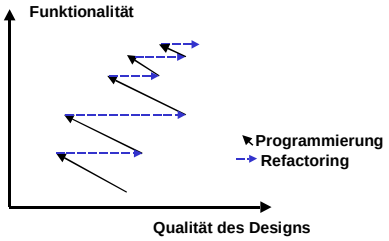

Abb. 3: Refactoring und normale Programmierung ergänzen sich

Zur Sicherung der Korrektheit semantikerhaltender Transformationen werden keine Verifikations-Techniken eingesetzt, sondern die vorhandene Test-Sammlung genutzt. Unter der Annahme der Existenz einer qualitativ hochwertigen Test-Überdeckung ist die Wahrscheinlichkeit hoch, dass fehlerhafte Modifikationen erkannt werden. Natürlich müssen oft Tests gemeinsam mit dem Code refaktorisiert werden, wodurch das Risiko entsteht, dass tatsächlich fehlerhafte Refactoring-Anwendungen nicht erkannt werden. Refactoring-Techniken zielen auf verschiedene Ebenen des Systems. Manche Refactoring-Regeln wirken im Kleinen, andere sind eher für Modifikation einer System-

Architektur geeignet. Durch die Möglichkeit, eine bereits im Code realisierte System-Architektur zu verbessern, verliert die Notwendigkeit a priori eine korrekte und stabile System-Architektur zur definieren an Brisanz. Natürlich sind Modifikationen an der System-Architektur kostenintensiv. In XP wird jedoch angenommen, dass die Wartung ungenutzter System-Funktionalität auf Dauer kostenintensiver ist. XP nimmt hier sehr deutlich den Standpunkt ein, die System-Architektur einfach zu halten und nur bei Bedarf durch geeignete Refactoring-Techniken zu modifizieren oder zu erweitern.

4 XP in Lehre und Forschung

Die aktuelle Lehrmeinung im Software-Engineering fordert eine saubere Vorgehensweise von der Analyse der Geschäftsvorfälle bis zur Wartung. Es wird eine systematische, teilweise bürokratisierte Vorgehensweise mit entsprechender Dokumentationsleistung erwartet. XP bricht also mit dieser insbesondere in Europa vorherrschenden Lehrmeinung. Weil aber XP eine deutliche Praxisrelevanz bekommen hat, ist es für die akademische Welt geboten, sich auch mit dieser Methodik kritisch auseinander zu setzen. Bei genauerer Analyse wird zu erwarten sein, dass die akademische Lehrmeinung und Extreme Programming nicht unversöhnlich sind. So sind z. B. die Techniken des Refactoring durchaus eine pragmatische Umsetzung eines uralten akademischen Wunsches nach transformationeller, systematischer Softwareentwicklung. Es ist Aufgabe der akademischen Welt, die einzelnen Techniken von XP kritisch zu durchleuchten, seine Annahmen und Schlussfolgerungen mit systematisch erhobenem Zahlenmaterial zu be- oder widerlegen und auf dieser Basis konstruktive Verbesserungsvorschläge vorzunehmen. Folgender Abschnitt beschreibt dazu einige Ansatzpunkte. Zuvor jedoch soll aufgezeigt werde, wie sich XP-Elemente für die Lehre einsetzen lassen.

4.1 XP in der Lehre

Während eines Studiums lösen Studenten zahlreiche kleine Programmieraufgaben. Darüber hinaus führen sie typischerweise zwei bis drei halbjährige Systementwicklungsprojekte durch, die ernsthafteren Projektcharakter simulieren. Wird ein solches Projekt in Teamarbeit durchgeführt, so können einige Elemente von XP genutzt werden, um die Effizienz der Beteiligten und die Qualität des Ergebnisses zu steigern. So erlaubt etwa die Forderung nach automatisierten Tests eine verbesserte Übergabe von Ergebnissen. Ein permanent anwesender Kunde (Aufgabensteller, Betreuer), der aktiv an der Systementwicklung mitwirkt, kann als Experte und Vorbild für die Studenten wirken. Die Verinnerlichung fundamentaler Prinzipien, wie Einfachheit, inkrementelle Änderungen oder schnelles Feedback hilft den Studenten in späteren Projekten.

Um die Notwendigkeit von Tests zu motivieren, können diese bereits im ersten Semester anhand kleiner Programmieraufgaben eingesetzt werden. So kann eine Test-Sammlung vorgegeben sein, die den Studenten die selbständige Überprüfbarkeit ihrer Lösung ermöglicht. (eine ausführlichere Test-Suite kann dann für die Betreuer zur Korrektur eingesetzt werden). Alternativ kann eine leicht fehlerhafte Lösung vorgegeben sein, die durch eine Test-Suite zu korrigieren ist. Danach sollten die Studenten motiviert sein, Programmierlösungen selbständig mit Test-Suite abzuliefern.

Ein weiteres lehrreiches Element von XP ist die sogenannte Extreme Hour (XH). In sieben Phasen zu je 10 Minuten werden die Rollen des XP-Prozesses, Iterationsplanung,

Release-Planung, Priorisierung der Anforderungen, Aufwandsschätzungen und ähnliches mehr geübt. Der spielerische Zugang zur Vorgehensweise von XP ist bei Studenten sehr beliebt und gleichzeitig lehrreich.

4.2 Wissenschaftliche Aufarbeitung von XP

Ziel einer Auseinandersetzung mit XP ist sicher die konstruktive Verbesserung von XP, die Integration mit anderen existierenden Techniken oder deren fundierte Ablehnung. XP wurde erstmals Mitte des letzten Jahrzehntes publiziert, ist aber erst Ende 1999 populär geworden. Entsprechend ist eine fundierte Basis von Zahlenmaterial zum Thema XP noch nicht vorhanden. Die spezifische Problematik der Erhebung von Zahlenmaterial bei Software-Entwicklungsprojekten ist bei XP in der gleichen Weise gegeben. Es wird also einige Zeit dauern, bis über gefühlsbasierte Indikatoren hinaus ein besseres Verständnis über Vor- und Nachteile von XP entwickelt sein wird. Eine gesicherte Basis an erfolgreichen und an erfolglosen XP-Projekten existiert derzeit nicht. Indikatoren für die Ursachen gescheiterter XP-Projekte gibt es nicht. Untersuchungen weshalb XP-Projekte erfolgreich abgelaufen sind gibt es ebenfalls nicht. Insbesondere wäre zu erwarten, dass alleine aufgrund hoher Motivation, die die Verwendung eines neuen Softwareentwicklungs-Prozesses mit sich bringt, die Erfolgs-Wahrscheinlichkeit eines Projekts steigt. Dennoch gibt es zu einzelnen Aspekten von XP Arbeiten und Zahlenmaterial, die nicht direkt aus dem XP-Umfeld erhoben wurden. Ohne Anspruch auf Vollständigkeit sollen nachfolgend einige Aspekte von XP diskutiert werden.

Annahme von XP: Lineare Kostenkurve bei Fehlererkennung
Eine der Annahmen in XP stellt wesentliche Erkenntnisse klassischen Software-Engineerings in Frage: nämlich dass Kosten für Änderungen oder Fehlerbehebung exponentiell über die Projektlaufzeit steigen. Innerhalb der XP-Gemeinde wird die Kostenkurve von Entwicklungsfehlern derzeit kontrovers diskutiert. Durch die Nutzung besserer Sprachen (Java), besserer Datenbanktechnologie und die Verbesserung der Programmierpraktiken, insbesondere auf Basis von Kodierungs-Standards, sowie bessere Werkzeuge und Entwicklungsumgebungen, wird behauptet, dass Fehler-Kosten nicht mehr exponentiell über die Zeit steigen [Be99, Kapitel 5]. Obwohl gewisse Argumente zumindest partiell für diese These sprechen, muss ein Beleg dieser Aussage durch Zahlenmaterial erst erfolgen. Die Validierung dieser These ist jedoch essentiell für XP.

Annahme von XP: Viele Änderungswünsche der Kunden
Des weiteren nimmt XP an, dass Software generell stark häufig Änderungswünschen der Kunden unterliegt. Die dadurch ausgelöste permanente Erosion der in normalen Softwareentwicklungsprozessen entstehenden Dokumentation ist dann konsequenterweise durch XP-Techniken zu entgegnen. Bei betrieblichen Software-Systemen, sowie insbesondere internetbasierter Software (e-Business) kann diese Annahme sicherlich als richtig erachtet werden. Speziell im e-Business ist Time-to-Market genauso wichtig wie die Qualität der Software. Es gibt jedoch auch andere Projektumfelder: z. B. im Bereich eingebetteter Systeme, bei Projektaufträgen der öffentlichen Hand, verteilt realisierten Projekten oder Projekten mit deutlich mehr als 10 Entwicklern.

Annahme von XP: Einarbeitung in dokumentierten Code ist wenig aufwendig
Eine weitere Annahme von XP ist, dass die Einarbeitung in gut dokumentierten Code, gegebenenfalls unter Zuhilfenahme eines geeigneten Reengineering-Werkzeugs nicht

wesentlich aufwendiger ist, als der Zugang über die Dokumentation. Zusätzlich bietet ein Zugang über den Code die Sicherheit, keine falsche Information zu nutzen. Dies ist heute vor allem ein Problem nicht genügend ausgereifter Software-Engineering-Werkzeuge, die die Konsistenz zwischen Modellen und Code nicht ausreichend sichern können. Dem verbreiteten Ansatz der Zuordnung von Modulen an einzelne Entwickler wird in XP eine Absage erteilt. Stattdessen dürfen und sollen alle Entwickler von allen Modulen Kenntnisse und das Recht zur Modifikation haben. Ist schlechter Code erkannt, so soll er verbessert werden. Dies führt leider häufig zu Befindlichkeiten zwischen den Projektbeteiligten, kann aber durch paarweise Programmierung deutlich reduziert werden. Kritisch ist auch zu bewerten, dass Individualisten mit unterschiedlichen Auffassungen sich gegenseitig die Code-Struktur zerstören können. Mit detaillierten Kodierungs-Standards und einer ausgeprägten teamorientierten Projekt-Kultur lässt sich dem begegnen.

Pair-Programming

Die paarweise Programmierung (Pair-Programming) greift stark in die Projekt-Kultur ein. Pair-Programming bedeutet, dass zwei Personen an einem Rechner abwechselnd die Tastatur bedienen, während der jeweils andere über die Schulter sieht. Rein rechnerisch gibt das zunächst doppelten Personalaufwand. Durch geeignete Projekte, speziell auch durch Studenten, können jedoch die Auswirkungen des Pair-Programmings relativ gut untersucht werden. Bei solchen Untersuchungen wurde bereits festgestellt, dass nach einer relativ kurzen Einarbeitungszeit in den neuen Programmierstil zwar der Gesamtaufwand gegenüber Einzelprogrammierung erhöht war, sich aber eine deutliche Reduktion der Projektdauer und insbesondere eine signifikante Erhöhung der Softwarequalität ergeben hat [Wi00]. Um die Korrektheit dieser Ergebnisse zu validieren sind weitere Versuche durchzuführen.

Keine objektive Berichterstattung über XP-Projekte, Wartungsproblematik

Eine ganze Reihe von XP-Projekten haben mittlerweile Erfolgsmeldungen hervor gebracht. Die Dunkelziffer erfolgloser XP-Projekte ist unbekannt. Des weiteren ist unklar, wie sich XP-Projekte auf die Wartungsproblematik auswirken. Ein System das längere Zeit ruht „erodiert“. Erosion geschieht aus zwei Gründen: zum einen verändern sich die Anforderungen der Anwender, teilweise getrieben durch veränderte Geschäftsprozesse, teilweise durch Verbesserung des Verständnis des Machbaren. Zum anderen erodiert das Wissen über die Software in den Köpfen der Entwickler. Dadurch wird die Wartungsproblematik und die Wiederaufnahme einer Weiterentwicklung immens erschwert. Aufgrund der Neuheit des XP-Prozesses ist derzeit kein Zahlenmaterial zur Frage der Wartungsfreundlichkeit von XP-Ergebnissen vorhanden. Hier werden Langzeit-Untersuchungen notwendig sein.

Testmethodik für XP

Obwohl Tests in XP eine außerordentliche Rolle spielen, wird nicht auf fundierte Testmethodiken und Testwerkzeuge zurück gegriffen. Stattdessen wird explizit vorgeschlagen, das einfachste vorhandene Testwerkzeug zu verwenden [JAH00, s.S.105]. Eine verstärkte Integration der vorhandenen Testkonzepte, insbesondere von Testmetriken und Testgenerierungs-Werkzeugen ist sicherlich wünschenswert. Techniken zur Messung der Systemdefekte auf Basis vorhandener Testsammlungen wären ideale XP-Analyse-Werkzeuge.

Refactoring
Refactoring ist eine der interessantesten Techniken, die mit XP populär werden. Die Evolution eines Softwaresystems durch kleine, systematisch angewendete Transformationstechniken auf Code-Basis bietet ein weites Forschungsfeld. Neben der Entwicklung geeigneter Werkzeuge zur automatischen oder halbautomatischen Transformation von Code verschiedener Programmiersprachen sind insbesondere Werkzeuge zur automatisierten oder halbautomatischen Verifikation der Korrektheit von Transformationen interessant. Im Compilerbau werden seit langem Transformationen zur Codeoptimierung eingesetzt. Techniken zur Optimierung der Code-Struktur (Ausbalancierung von Aufruf- oder Klassen-Hierarchien, Faktorisierung gemeinsamer Codestücke in eigenständige Methoden oder Entfernung nicht genützten Codes) könnten sich in diesem Kontext als hilfreich erweisen. Eine Ausweitung der Transformationstechniken auf weitere Notationen, wie z. B. UML Klassendiagramme, und deren Integration mit dem Code in geeigneten Werkzeugen dürfte ebenfalls sinnvoll sein.

5 Ergänzungen zur XP

XP und Frameworks
Wesentliche Elemente von XP sind die Einfachheit des Entwurfs, die stete Entfernung nicht genützten Codes und die Realisierung genau der Funktionalität, die von den Kunden gewünscht ist. Dementsprechend ist XP für die Entwicklung eines Frameworks ungeeignet. Das heißt aber nicht, dass in XP-Projekten keine existenten Frameworks eingesetzt werden sollten oder dass nicht „zufällig“ durch mehrere verwandte XP-Projekte ein Framework entstehen kann. Interessant ist XP vor allem beim Einsatz von Frameworks wenn diese geeignet vorbereitet sind. Ein Framework definiert sich durch eine Sammlung von Klassen mit starker Interaktion, eigenem Kontrollfluss und eine Menge von Klassen bzw. Methoden, die dazu gedacht sind, durch applikationsspezifische Implementierungen redefiniert zu werden. Die Verwendung eines Frameworks ist im Sinne von XP erwünscht, wenn es das Projekt schneller zu einem qualitativ hochwertigeren Ergebnis bringt. Schwierig jedoch wird es, wenn einzelne Verhaltensweisen des Frameworks nicht bekannt sind. Durch die Entwicklung von Tests können diese Verhaltensmuster in Erfahrung gebracht werden. Umgekehrt kann und sollte ein Framework selbst Testklassen mitliefern, die dazu geeignet sind, Applikationsklassen auf ihre Korrektheit und Robustheit im Bezug auf die Redefinition von Hot-Spots zu prüfen. Solche Framework-Tests können für die Applikations-Entwicklung sehr hilfreich sein. Leider ist dem Autor bis dato kein Framework bekannt, das eine derartige Test-Sammlung mitliefern würde.

Extreme Modeling
Eine Weiterentwicklung des Extreme Programming, ist das sogenannte „Extreme Modeling“. Es basiert auf der Überlegung, dass die Entwicklung noch effizienter und in noch kürzeren Zyklen ablaufen kann, wenn statt einer klassischen Programmiersprache eine ausführbare Teilsprache der UML verwendet wird. Mehrere Werkzeug-Hersteller, wie z.B. GentleWare.de entwickeln derzeit solche Code- und Testgeneratoren. Ziel ist es, Klassen-, Sequenz-, Statechart- und Use-Case-Diagramme teilweise zur konstruktiven Generierung von Code und teilweise zur Generierung von automatisierten Tests einzusetzen. Des weiteren wird über UML-basiertes Refactoring wieder nachgedacht, denn Refactoring wurde ursprünglich für Klassendiagramme eingeführt. Es ist vorstellbar,

dass in wenigen Jahren eine konsolidierte, ausführbare Teilsprache der UML existiert, die in Kombination mit textuellen Anteilen aus Java oder einer anderen Programmiersprache ein ideales Programmierwerkzeug für XP-basierte Projekte darstellt. Bei all den Vorteilen die Extreme Modeling auf UML-Basis haben kann, sollte jedoch darauf geachtet werden, dass UML nicht nur als Programmiersprache, sondern auch als abstrakte Modellierungssprache für frühe Projektaktivitäten entwickelt wurde. Dies widerspricht aber keineswegs der Nutzung einer ausgezeichneten Teilsprache zur Code-Generierung.

Hierarchische Organisation mehrerer XP-Projekte

Aus der betriebswirtschaftlichen Projektorganisation kommt ein Vorschlag, um XP-Projekte mit deutlich mehr als 10 Mitarbeitern umsetzen zu können [JR01]. Wesentliche Idee dabei ist, die Strukturierung großer Projekte in viele kleine XP-Projekte und einen Steuerkreis (siehe Abb. 4). Der Steuerkreis setzt sich zusammen aus wenigstens einem Mitglied jedes Teilprojekts, einem Werkzeug-Verantwortlichen, einem Versions-Manager, dem Gesamtprojektleiter und mindestens einem Kunden. Unter anderem identifiziert und bearbeitet er auftretende Risiken, entscheidet über Technologiefragen, identifiziert und rearrangiert Teilprojekte, sichert den Kommunkationsfluss zwischen den Teilprojekten und definiert deren softwaretechnische Schnittstellen. Gerade in der Schnittstellen-Problematik liegt aber die große Schwierigkeit der Zerteilung eines großen Projekts in XP-Teilprojekte. Hier müssen im Vorfeld bestimmte Aktivitäten zur Definition von Schnittstellen und meist der dahinterliegenden Software-Architektur getroffen werden. Dies ist z. B. in den Bereichen eingebettete Systeme und Telekommunikation zumeist der Fall. Projektorganisation kann dann eine wie in Abb. 4, Mitte dargestellte Form annehmen.

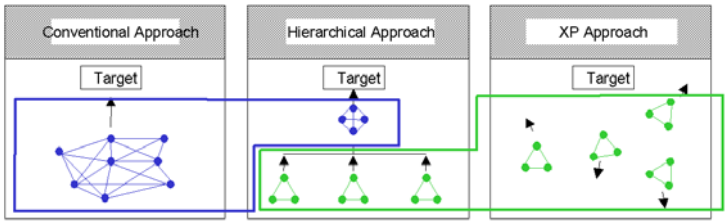

Abb. 4: Hierarchische XP-Organisation mit projektübergreifendem Steuerkreis (mitte) im Vergleich zu konventionellem Ansatz (links) und reinem XP-Ansatz (rechts)

6 Schluss-Statement

Extreme Programming ist eine leichgewichtige Softwareentwicklungs-Methode, mit einem Wertesystem sowie einer Sammlung von Grundprinzipien und Entwicklungspraktiken. Neuartig an XP ist die veränderte Gewichtung einzelner Konzepte und insbesondere der explizite Verzicht auf bestimmte wesentliche Tätigkeiten klassischer Projektorganisation. So führt z. B. der Verzicht auf Dokumentation zu einer veränderten Projektkultur mit intensiver Kommunikation der Entwickler untereinander und enger Einbeziehung des Kunden.

Trotz des extremen Namens ist XP nicht so radikal wie es den Anschein hat. Es ist z. B. möglich und in der Praxis sinnvoll, einzelne Konzepte der XP-Vorgehensweise in eine vorhandene Projektkultur einzubauen. Insbesondere die stärkere Betonung der automatisierten Tests und des Refactorings bieten sich an. XP führt also keineswegs zurück zu

den Anfängen der Software Engineering-Kultur, sondern sucht auf Basis der vorgestellten „best practices“ manchen Ballast heutiger Vorgehensmodelle zu vermeiden.

Nach der Meinung des Autors gehören XP und seine Techniken ins Portfolio eines Softwareentwicklers, um sich derer bei Bedarf zu bedienen. XP kann gleichberechtigt neben anderen Software-Engineering-Techniken dem Entwickler als Handwerkszeug zur Verfügung stehen. Weil XP außerdem für die Lehre gewisse Vorteile besitzt, vertritt der Autor die Meinung, dass XP in einer geeigneten Vorlesung oder besser noch einem Praktikum den Studenten nahe gebracht werden darf.

XP stellt in seiner heutigen Form nur einen Zwischenzustand dar und wird sich weiter entwickeln. Dazu gehören bessere Werkzeugunterstützung für die Softwarekonstruktion und –analyse, genauso wie die Entwicklung eines besseren Verständnisses über die Vor- und Nachteile von XP auf Basis wissenschaftlich gesicherter Untersuchungen.

Danksagung

Diese Arbeit wurde unterstützt von der Bayerisches Staatsministerium für Wissenschaft, Forschung und Kunst durch den Bayerischen Habilitationsförderpreis und FORSOFT II. Mein Dank gilt der Firma ESG für die gemeinsame Durchführung von XP-Projekten, Andreas Günzler, Wolfgang Schwerin und unseren Studenten für die Bereitschaft, XP zu testen.

Literaturverzeichnis

[Ba85] F. L. Bauer, R. Berghammer, M. Broy, W. Dosch, F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse, B. Krieg-Brückner, A. Laut, T. Matzner, B. Möller, F. Nickl, H. Partsch, P. Pepper, K. Samelson, M. Wirsing, H. Wössner: The Munich Project CIP. Volume I: The Wide Spectrum Language CIP-L. LNCS 183. Berlin, Springer 1985.

[Be99] Beck, K. : Extreme Programming Explained. Addison-Wesley, 1999

[BF00] Beck, K., Fowler, M.: Planning Extreme Programming. Addison-Wesley, 2000

[Cu01] Cunningham, W.: Extreme Programming Roadmap. http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap, 2001

[Ec00] Eckstein J.: XP - Extreme Programming. In: basicpro, 3/2000

[EH00] Elting, A., Huber, W.: Immer im Plan? Programmieren zwischen Chaos und Planwirtschaft. In: c’t., Heise Verlag, 2/2000

[EH01] Elting, A., Huber, W.: Vorgehensmodelle contra Extreme Programming. 2 teilig. In: sw development. Magazin für Software-Entwicklung. E.S.T.! Verlag, 1&2/2001

[Fo99] Fowler, M: Refactoring. Addison-Wesley, 1999

[JAH00] Jeffries, R., Anderson, A., Hendrickson, C.: Extreme Programming Installed. Addison-Wesley, 2000

[Je01] Jeffries, R.: www.xprogramming.com, 2001

[JR01] Jacobi, C., Rumpe, B. : Hierarchical XP – Improving XP for large scale projects. In: XP’2000 Conference Proceedings. Addison-Wesley, 2001

[JUnit01] Junit Test-Frameworks für mehrere Sprachen, www.junit.org, 2001

[Kr00] Kruchten, P.: The Rational Unified Process. An Introduction. Second Edition, Addison-Wesley, 2000

[Ve00] Versteegen, G. (Hrsg.): Das V-Modell in der Praxis. dpunkt.verlag, 2000

[We99] Wegener, H.: Extreme Ansichten. Für und Wider des Extreme Programming, In: iX Journal. Heise Verlag, 12/1999

[We01] Wells, D.: Extreme Programming, http://www.extremeprogramming.org/, 2001

[Wi00] Williams L., Kessler R., Cunnigham W., Jeffries R.: Strengthening the Case for Pair Programming. In: IEEE Software 17/4, 2000