Rollenveränderung in der Agilen Software-Entwicklung: Das „Projektmanagement-Labor“ der BA Lörrach

Eckhart Hanser

Studiengang Angewandte Informatik
Berufsakademie (BA) Lörrach – University of Cooperative Education
Hangstraße 46-50
79539 Lörrach
hanser@ba-loerrach.de

Abstract: Ausgehend von der Historie der Industrialisierung der Software-Entwicklung und den sich verändernden Anforderungen an modernes Software-Engineering gibt dieser Artikel einen Einblick in die wohl bekanntesten Agilen Programmiertechniken, Extreme Programming (XP) und Crystal Clear. Anhand praktischer Erfahrungen, die im IT-Labor der Berufsakademie Lörrach gesammelt wurden, werden einige der wichtigsten XP-Regeln u.a. Pair Programming hinterfragt und widerlegt. Am Beispiel von Crystal Clear wird gezeigt, wie der gelebte Prozess vom gewählten Prozessmodell abweicht. Ausgehend von den Ergebnissen des IT-Labors werden die wichtigsten "Zutaten" formuliert, die für eine erfolgreiche Software-Entwicklung im Team notwendig sind.

1 Einleitung: Kurze Historie der Software-Entwicklung

In den 70er und 80er-Jahren des vergangenen Jahrhunderts war die Welt der Software-Entwickler noch „in Ordnung“. Nach einem Prozess der Industrialisierung der Software-Entwicklung stand am Anfang eines Projekts eine ausführliche schriftliche Spezifikation, und man konnte davon ausgehen, dass sich diese selbst bei mehrjährigen Projekten bis zum Schluss kaum änderte. Man entwickelte gemäß dem sog. Wasserfall-Modell, was eine Projektentwicklung von der Spezifikation zum Design, und dann weiter über die eigentliche Programmierung (Implementation) bis zum Test zuließ – ähnlich den Kaskaden eines Wasserfalls.

Das Umfeld von Software-Projekten veränderte sich in den 90er-Jahren rasant. Projekte wurden immer kurzlebiger, die Anforderungen änderten sich immer mehr schon während der Projektlaufzeit und oftmals konnte der Kunde am Anfang des Projekts noch keine detaillierten Aussagen zum Umfang des Gesamtprojekts machen. Gleichzeitig wurde die durchschnittliche Entwicklungs-Teamgröße immer kleiner. Diese Entwicklung hatte einen Höhepunkt im New Economy Hype, der in einem Buch von Lothar Späth [Sp01] als „Revolution“ gepriesen und mit der rasanten Entwicklung des Internets gleichgesetzt wurde.

Besonders Software-Entwickler, die nicht in Nischen wie der militärischen oder pharmazeutischen Entwicklung arbeiteten, waren folglich gezwungen, das bis dahin verwendete „industrielle“ Wasserfall-Prozessmodell durch geeignetere Software-Entwicklungsmodelle zu ersetzen. Ansätze, die für Furore in der Szene gesorgt haben, sind das 1996 von Kent Beck entwickelte Extreme Programming, sowie Crystal von Alistair Cockburn (2004), so genannte „Agile“ Prozessmodelle.

2 Das Agile Manifest

Diese neuen Prozessmodelle sind die Reaktion auf die so genannten „schwergewichtigen“ Prozess- bzw. Vorgehensmodelle, wie den Unified Process (UP) [JBR99] von Rational (jetzt IBM) oder das V-Modell [Ra06], das Vorgehensmodell der deutschen Bundesbehörden. Obwohl beide Ansätze sehr unterschiedlich sind – UP beleuchtet den iterativ-inkrementellen Software-Entwicklungsprozess, während das V-Modell die Sicht des Kunden auf das Software-Projekt in den Vordergrund stellt – gelten beide als sehr „dokumentenlastig“, was zumindest bei „Anfängern“ erheblichen zusätzlichen Aufwand bedeutet.

Branchen wie etwa die militärische oder pharmazeutische Software-Entwicklung, in denen Gefahr für Leib und Leben bestehen kann, können nicht auf ausführliche Dokumentation verzichten. Das verlangen zu Recht die zuständigen Behörden. Es handelt sich jedoch um eng abgegrenzte Felder. In der Mehrzahl der Software-Projekte jedoch, sollte die Entwicklung funktionierender Software im Mittelpunkt stehen, wie Kritiker wie Ward Cunningham, Martin Fowler, Alistair Cockburn oder eben Kent Beck betonen. Dies führte 2001 zur Verkündung des „Manifesto for Agile Software Development“, dem Agilen Manifest [Am01], welches die Vorgenannten und 13 andere unterzeichneten. Leitsätze des mittlerweile weltweit unterstützten Agilen Manifests sind:

- Die individuellen Beteiligten und ihre Interaktion sind wichtiger als Prozesse und Werkzeuge.
- Die Ablieferung lauffähiger Software ist wichtiger als umfassende Dokumentation.
- Die Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen.
- Auf Änderungen der Anforderungen einzugehen ist wichtiger als die sture Verfolgung eines Plans.

3 Extreme Programming (XP)

Kent Beck entwickelte sein Prozessmodell *Extreme Programming* [Be00] 1996 im Rahmen des Chrysler-Projekts *C3 payroll*, das die Entwicklung einer Lohn- und Gehaltslisten-Software zum Ziel hatte. Das Projekt war zum Zeitpunkt des Eintretens von Beck bereits so gut wie gescheitert. Man war also bereit, auf neue Ansätze zu hören und gab somit Beck die Möglichkeit, die zuerst informelle Praxis als neues Prozessmodell zu formulieren und das Smalltalk-Projekt neu zu starten. Nach Becks eigener Aussage war das Projekt ideal für seinen neuen Ansatz,

- weil sich die Spezifikationen schnell änderten,
- weil der Kunde keine endgültige Vorstellung vom Funktionsumfang hatte,
- weil eine hohe Produktivität der Programmierer gefordert war,
- weil das Projekt ein hohes (auch zeitliches) Risiko in sich trug,
- weil die Projektgruppe inkl. Managern, Programmierern und Kunden eine Stärke von weniger als 12 Mitarbeitern hatte
- und weil die zu erstellende Software automatisiertes Testen erlaubte.

In der ersten Phase war C3 sehr erfolgreich und wurde 1997 produktiv. Somit wurde C3 zum „Flaggschiff“ für XP.

Im Gegensatz zu anderen Verfechtern des Agilen Manifests, wie z. B. Alistair Cockburn, dem Begründer des Crystal-Prozessmodells [Co04], schreibt Kent Beck die Vorgehensweisen im Projekt strikt vor. Er definiert „Rules and Practices“ zu Planung, Design, Kodierung und Testen, auf deren Einhaltung er besteht. Über allen Regeln stehen die 4 Schlüsselbegriffe: *Kommunikation, Einfachheit, Feedback und Mut*. Kommunikation zwischen den Team-Mitgliedern, aber auch mit dem Kunden soll ausführliche Dokumentation ersetzen. Ein einfaches Design soll die Fehleranfälligkeit des Systems reduzieren, Feedback speziell des Kunden soll das Projekt in die richtige Richtung treiben. Das Team soll aber auch den Mut haben, XP abzuändern, wenn gewisse Aspekte im speziellen Projekt nicht funktionieren („Fix XP when it breaks.“).

Es fällt auf, dass diese Regeln und Verhaltensweisen meist nicht neu sind. Es ist eher ihre Kombination und ihre strikte Einhaltung, was XP ausmacht. Man kann die XP-Regeln durchaus in 3 Kategorien einteilen: Regeln, die jeder moderne Projektmanager sofort akzeptiert und meist auch schon einsetzt, Regeln, die einleuchten als Alternative zu eigenen Vorgehensweisen und Regeln, die Widerspruch hervorrufen:

Kein erfahrener und zeitgemäßer Projektleiter bezweifelt die Wichtigkeit einer vernünftigen Versions- und Release-Planung, kleiner Releases, einer iterativ-inkrementellen Vorgehensweise und eines einfachen Designs mit Namenskonventionen. Ebenso gehört heutzutage neben Prototyping ein ausführliches und automatisiertes Testen „ab dem ersten Tag“ bereits auf Modul-Ebene zu den modernen Software-Projekttechniken. Ein erfahrener Projektleiter misst die „Geschwindigkeit“ seines Projekts und achtet darauf, dass die Mitarbeiter möglichst keine Überstunden machen (müssen).

Jeder Projektleiter wird „Stand-Up Meetings“ also kurze morgendliche Sitzungen im Stehen als effizientes Kommunikationsmittel schätzen lernen. Ebenso erhöht ein häufiges Verbessern des Designs, ein sog. *Refactoring*, die Qualität der Software und ist mit entsprechender Tool-Unterstützung machbar (*Computer Aided Software Engineering, CASE* [Ba98]). Schließlich gibt Kent Beck die ausdrückliche Erlaubnis, XP anzupassen, wenn das Projekt es erfordert. Dies gibt dem erfahrenen Projektleiter die Sicherheit, den Prozess den praktischen Anforderungen unterordnen zu können und nicht umgekehrt.

Die dritte Gruppe der XP-Regeln gibt mehr Anlass zu Diskussionen, wie der Autor aus eigener Erfahrung weiß, und soll in diesem Artikel näher beleuchtet werden:

(1) Beck fordert die permanente Mitgliedschaft des *Kunden im Team*. Dies soll eine schnelle Reaktion auf sich ändernde Kundenwünsche ermöglichen, selbst in späten Projektphasen. Spezifikationsdetails können „face to face“ mit dem Kunden besprochen werden. Die geforderte dauernde Anwesenheit des Kunden im Projekt ist in der Praxis naturgemäß ein Problem, da erfahrene Kunden kaum vollständig einem Software-Projekt zur Verfügung stehen. Ein anderer Einwand ist, dass der Kunde auf diese Weise zu viele Interna des Projekts (auch gemachte Fehler!) mitbekommen könnte. Würde dadurch nicht sein Vertrauen in die Qualität der Entwickler-Mannschaft sinken? Und wird dadurch die Projekt-Kommunikation insbesondere mit dem Kunden wirklich besser?

(2) *User Stories*: Die Anforderungen sollen nicht in Form von Dokumenten, sondern in Form von kurzen, dreisätzigen Mini-Spezifikationen vom Kunden in seiner „Sprache“ abgefasst werden. Der Aufwand für die Implementierung des zugehörigen Software-Moduls soll 3 Wochen nicht überschreiten. Kann der Kunde das? Wie kann er 3 Wochen Aufwand abschätzen? Macht eine solche zeitlich feine Gliederung überhaupt Sinn? Braucht er Unterstützung (von wem)?

Schließlich die „berühmtesten“ XP-Regeln:

(3) Programmieren in Paaren – *Pair Programming*: 2 Programmierer arbeiten zusammen an einem Computer. Beck verspricht eine Erhöhung der Software-Qualität bei gleicher Effizienz wie bei einem Team zweier Programmierer an 2 Computern. Im Paar soll ein Programmierer taktisch denken, sich also mit dem Funktionsumfang einer aktuellen Klassenmethode beschäftigen, während der andere strategisch denkt, sich also mit der Frage beschäftigt, ob obige Klassenmethode überhaupt in diese Klasse passt. Die Programmierer wechseln regelmäßig (ggf. alle paar Stunden) ihre Rollen. Diese Forderung wird wohl in den wenigsten Projekten erfüllt. Die Frage nach dem Sinn dieser XP-Regel soll im Weiteren untersucht werden.

(4) *Collective Code Ownership, move people around*: Jedes Team-Mitglied soll Verantwortung für den gesamten Code tragen und diesen auch entsprechend verstehen. Jeder, der eine Änderung am Code eines anderen Paares braucht, kann diese selbständig durchführen, da das automatisierte Testen mit einer entsprechenden Versionskontrolle es auf eine Art ermöglicht, den „Urzustand" der Software vor der Änderung wieder herbeizuführen. Dieser Punkt ist heikel. Jeder der schon einmal mit entsprechenden Tools gearbeitet hat, weiß, dass man hierzu ein tiefes Verständnis der zu ändernden Software haben muss oder die Bereitschaft zum Erlangen dieses Verständnis mitbringen muss. Auch diese Regel soll untersucht werden.

(5) *Integration in kurzen Abständen*, idealerweise kontinuierlich, ausgehend von den Paaren, die eigenverantwortlich nacheinander an einem Integrationsrechner die Projektmodule zu einem lauffähigen Release verknüpfen. Hier muss die Frage geklärt werden, ob alle Paare genügend Willen und Know-How mitbringen um eine in der Regel komplexe Integration erfolgreich durchzuführen.

Außerdem soll der Frage nachgegangen werden, ob die klassischen Rollen des Projektleiters und des Qualitätsmanagers in einer agilen Umgebung noch Sinn machen, bzw. wie sie sich ändern.

4 Das IT-Labor der Berufsakademie Lörrach

Das IT-Labor ist eine Veranstaltung im 6. Semester des IT-Studiums, in dem der jeweilige Kurs gemeinsam als Team ein Software-Projekt durchführt. Jeden Frühling wird an 11 Tagen zu jeweils 5 Kursstunden mit ca. 20 Team-Mitgliedern an diesem Projekt gearbeitet. Dies ergibt pro Semester effektiv ca. 800 Projekt-Stunden. Naturgemäß ergibt sich die Zahl der Team-Mitglieder aus der Zahl der Studierenden plus Dozent. Das IT-Labor besteht in dieser Form seit 2004. Der Autor übernimmt dabei als Dozent die Rolle des Kunden (und ab und zu des Beraters), die Rollen Projektleiter und Qualitätsmanager werden entweder von ihm bestimmt oder vom Team gewählt [Ha06].

Neben dem vordergründigen Ziel, eine den Spezifikationen entsprechende stabile Software zu entwickeln, werden jedoch auch tiefer gehende Fragen zur Projektorganisation beleuchtet:

(1) Gibt es XP Projektpraktiken, die nicht ohne Intervention eines (Projekt-) Managements funktionieren? Welche XP Praktiken sind kein Resultat der Selbstorganisation des Teams? Hier wird insbesondere die bereits erwähnte dritte Gruppe der XP-Regeln untersucht (s. III), die Anlass zu Diskussionen gibt.

(2) Praktische Erfahrung zeigt, dass Teams mit mehr als etwa 5 Mitgliedern sich in „Mini-Teams“ unterteilen:

- In XP wird ein solches Mini-Team „Paar“ genannt. Es besteht aus 2 Mitgliedern. Sind diese Paare in der Realität erfolgreich?
- Gibt es eine ideale Größe (größer 2) für erfolgreiche Mini-Team-Teams?
- Wie sieht die Mitgliederstruktur einen solchen (erfolgreichen) Mini-Teams aus?

Anlass zu dieser Fragenstellung geben neben den persönlichen Erfahrungen des Autors auch diverse Artikel in der Literatur. Martin Fowler weist beispielsweise daraufhin, dass das eingangs erwähnte Chrysler C3 Projekt, also das „Initialprojekt“ von Extreme Programming, nur in der ersten Phase bis 1997 ein Erfolg war. Im weiteren Projektverlauf geriet es immer mehr in Schwierigkeiten und wurde 1999 eingestellt. Somit ist bewiesen, dass auch XP kein Garant für einen erfolgreichen Projektverlauf ist [Fo05]. Auch Alistair Cockburn, dessen Crystal in der Basler Pharma eingesetzt wird, äußert sich kritisch. Bekannt ist in diesem Zusammenhang die XP-Karikatur auf seiner Web-Site, die 2 Programmierer vor einem Bildschirm zeigt und an Edvard Munchs Gemälde „Der Schrei“ erinnert [Co08].

5 Das Projekt

Im IT-Labor der BA Lörrach wird eine eMail-Marketing Software entwickelt. Aus einer unterliegenden Datenbank werden Newsletter-Inhalte ausgelesen und personalisiert an Abonnenten verschickt, die sich zuvor für den entsprechenden Newsletter angemeldet haben. Die Software verfügt über eine web-basierte grafische Benutzeroberfläche und ein Administrationsinterface. Die Abbildungen 1 und 2 zeigen zwei Eingabemasken des Newsletter-Systems. Das System ist mittlerweile nach ca. 3.000 Projektstunden funktionsfähig und mit Hilfe des Mono-Projekts auf Linux-Server portiert. Es soll aber auch funktionell noch erweitert werden. Ein „Inbound“-Handling, also die automatische Behandlung eintreffender Mails fehlt beispielsweise noch völlig. Es bleibt also noch genug zu tun für zukünftige Studenten.

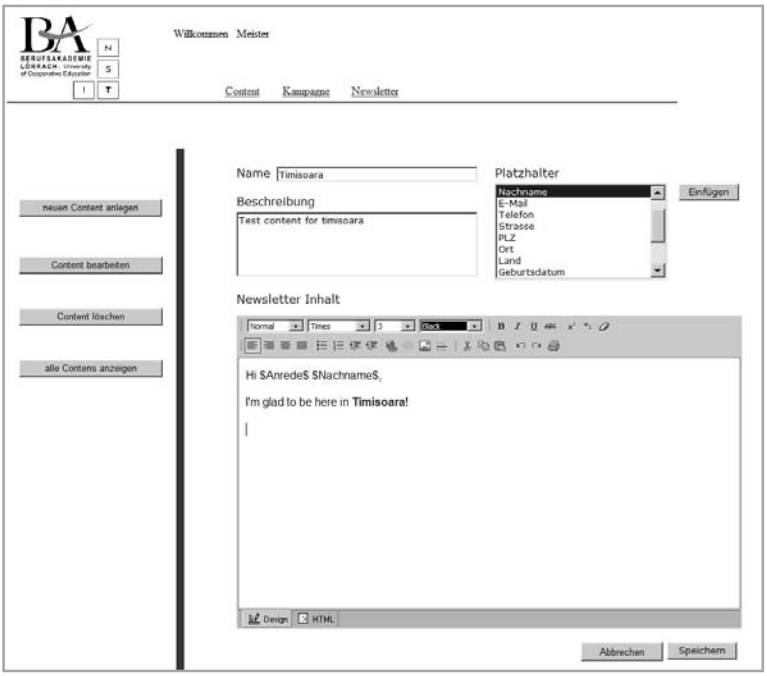
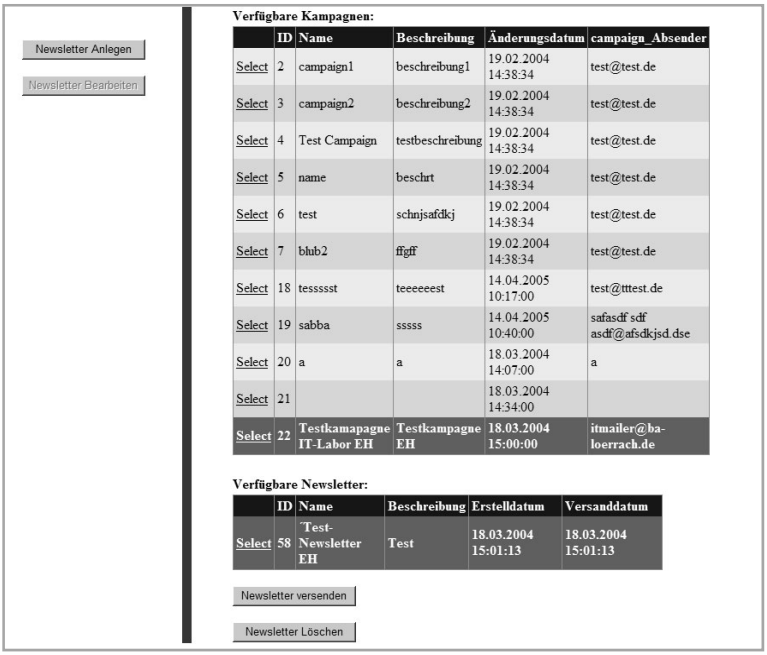


Abbildung 1: Newsletter-Editor



	ID	Name	Beschreibung	Änderungsdatum	campaign_Absender
Select	2	campaign1	beschreibung1	19.02.2004 14:38:34	test@test.de
Select	3	campaign2	beschreibung2	19.02.2004 14:38:34	test@test.de
Select	4	Test Campaign	testbeschreibung	19.02.2004 14:38:34	test@test.de
Select	5	name	beschrt	19.02.2004 14:38:34	test@test.de
Select	6	test	schnjsafdlkj	19.02.2004 14:38:34	test@test.de
Select	7	blub2	ffgff	19.02.2004 14:38:34	test@test.de
Select	18	tessssst	teeeeeest	14.04.2005 10:17:00	test@tttest.de
Select	19	sabba	sssss	14.04.2005 10:40:00	safasdf sdf asdf@afsdkjsd.dse
Select	20	a	a	18.03.2004 14:07:00	a
Select	21			18.03.2004 14:34:00	
Select	**22**	**Testkamapagne IT-Labor EH**	**Testkampagne EH**	**18.03.2004 15:00:00**	**itmailer@ba-loerrach.de**

	ID	Name	Beschreibung	Erstelldatum	Versanddatum
Select	**58**	**Test-Newsletter EH**	**Test**	**18.03.2004 15:01:13**	**18.03.2004 15:01:13**

Abbildung 2: Newsletter-Manager

6 Randbedingungen 2004 vs. 2005

Jede Session des IT-Labors wird vom Dozenten unter gewisse Randbedingungen gestellt. Somit können gezielt Fragestellungen beleuchtet werden, die sich aus dem Umgang mit Agilen Prozessmodellen ergeben. Diese Randbedingungen ergeben sich auch aus den Ergebnissen des vorigen Jahres. In der folgenden Tabelle sollen die Randbedingungen der Jahre 2004 und 2005 gegenübergestellt werden:

IT-Labor der BA Lörrach	**Session 2004**	**Session 2005**
Vorbedingungen	Der Dozent besetzt die Positionen Projektmanager und Qualitätsmanager mit geeigneten Studierenden.	Der Dozent besetzt die Positionen Projektmanager und Qualitätsmanager mit geeigneten Studierenden.
	Das restliche Team (Kurs) wird anfangs in Dreiergruppen (Mini-Teams) eingeteilt.	Team (Kurs) ist anfangs unstrukturiert, erhält aber den Auftrag, Mini-Teams in Form von XP-Paaren zu bilden.
	Die Teammitglieder haben Hintergrundwissen zum Thema Prozessmodelle, speziell zum Thema Agile Programmierung.	Die Teammitglieder haben Hintergrundwissen zum Thema Prozessmodelle, speziell zu Extreme Programming (XP).
	Der Kunde (der Dozent) ist Teammitglied.	Der Kunde (der Dozent) ist Teammitglied.
Entwicklungsplattform	Microsoft Visual Studio .NET 2003	Microsoft Visual Studio .NET 2003
Projektziel	Erfolgreiches Produkt, erfolgreiches Projekt.	Erfolgreiches Produkt, erfolgreiches Projekt.
	Business Logik und Grafische Benutzeroberfläche sollen von konkurrierenden Mini-Teams jeweils in C# und Visual Basic entwickelt.	Gemeinsame Implementierung eines voll einsatzfähigen eMail-Marketing-Systems in C#.
	Das Team soll sich einen geeigneten **Agilen Prozess wählen**.	**XP wird vom Dozenten gefordert.**
Erfolg der Session	Nur die Minimalanforderungen an Produkt und Prozess werden erfüllt.	Am Ende ein erfolgreiches Produkt und ein erfolgreicher Prozess.

Tabelle 1: Randbedingungen 2004 vs. 2005

Wie sind die unterschiedlichen Erfolge der beiden Sessions erklärbar? Was sind die notwendigen „Zutaten“ für ein erfolgreiches Projekt? Natürlich spielt die Stärke der Team-Mitglieder eine große Rolle, aber in beiden Jahrgängen waren starke Programmierer. Der Schlüssel zum Erfolg muss also auch maßgeblich in der Organisation des Teams liegen. Um dies zu untermauern sollen zunächst die Rollen des Projektmanagers und des Qualitätsmanagers beleuchtet werden. Machen diese klassischen Projektrollen in einer agilen Umgebung noch Sinn, bzw. wie ändern sie sich?

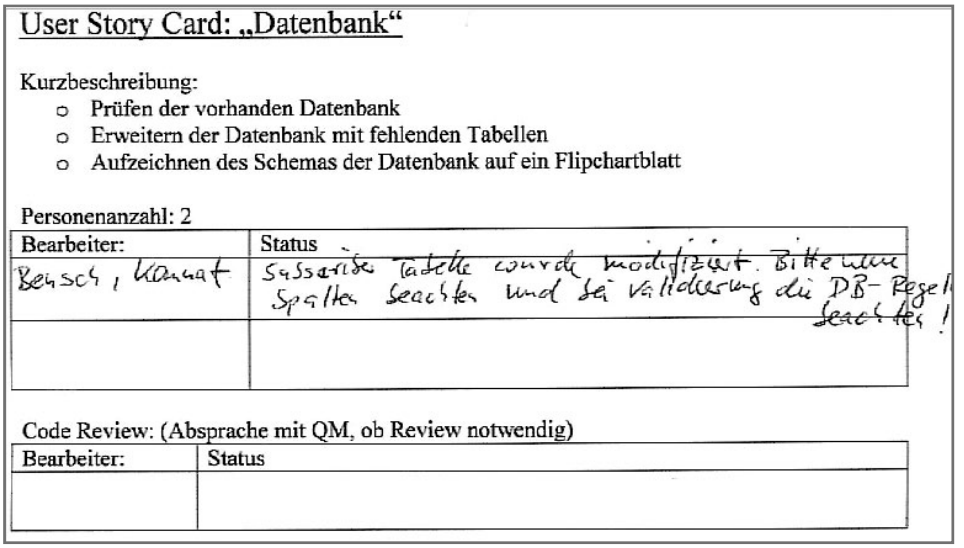
7 Projekt- und Qualitäts-Manager in Session 2005

Der „klassische“ Projektmanager führt sein Team und überwacht den Projektfortschritt. Er/sie erstellt das Projekthandbuch, welches das Projekt beschreibt, und den Projektplan für Aufwand, Zeit und Ressourcen. Der Projektmanager stellt die Weichen für ein erfolgreiches Projekt. In einem agilen Projekt ist diese klassische Projektrolle jedoch nicht eindeutig definiert. Also wurde beschlossen, dass ein Student diese Rolle übernehmen sollte um seine Rolle im agilen Team neu zu definieren.

Der Projektmanager 2005 beschrieb seine Rolle wie folgt [HBM06]:

- Koordinator des Projekts,
- Problemlöser,
- Kommunikator,
- .NET-Experte,
- Co-Designer von User Stories (Anwendungsfällen in Spezifikationen, Abbildungen 3 und 4).

Abbildung 3: User Story Card Board



User Story Card: „Datenbank“

Kurzbeschreibung:

- Prüfen der vorhanden Datenbank
- Erweitern der Datenbank mit fehlenden Tabellen
- Aufzeichnen des Schemas der Datenbank auf ein Flipchartblatt

Personenanzahl: 2

Bearbeiter:	Status
Beisch, Kunath	Sassarische Tabelle wurde modifiziert. Bitte neue Spalten beachten und bei Validierung die DB-Regel beachten!

Code Review: (Absprache mit QM, ob Review notwendig)

Bearbeiter:	Status

Abbildung 4: Details User Story Card

Die Rolle des Qualitätsmanagers, der im „klassischen“ Projekt die Projektqualität misst und verantwortlich ist für das Erstellen des Qualitätssicherungsplans, der Risikoanalyse und der Testpläne, ist im agilen Projekt nicht definiert. Wie im Fall des Projektmanagers wurde auch hier entschieden, dass ein Student die Rolle des Qualitätsmanagers übernehmen sollte um seine neue agile Rolle individuell zu definieren.

Der Qualitätsmanger 2005 beschrieb seine Rolle wie folgt [HBM06]:

- verantwortlich für Produkt- und Projektqualität,
  - definieren von Projektregeln (Abbildung 5),
  - durchführen von Reviews,
  - testen,
- Problemlöser,
- Kommunikator,
- .NET-Experte,
- Co-Designer von User Stories.

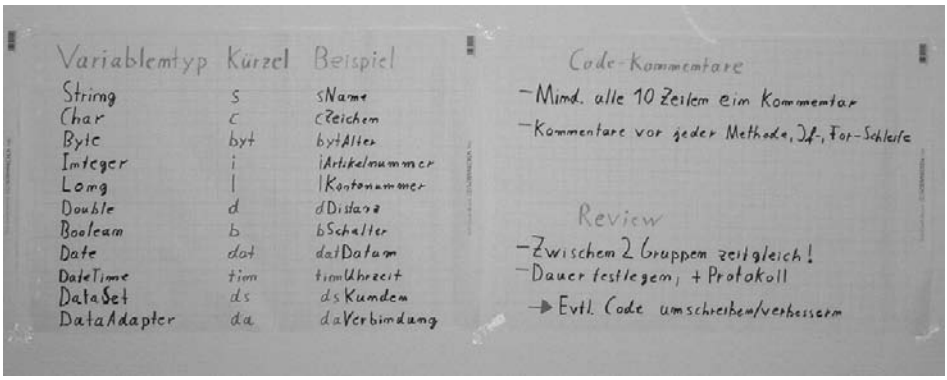


Abbildung 5: Projekt-Regeln

Es ist bemerkenswert, dass beide Studenten ihre neue Rolle sehr ähnlich definierten. Sie unterscheiden sich nur im ersten Punkt, also im „klassischen Teil“ ihrer neuen agilen Projektrolle. Die meiste Zeit übten beide Studenten ähnliche Tätigkeiten aus: Sie brachten dem Team (agile) Projektregeln und unterstützten es bei der Durchführung, insbesondere bei der Kommunikation. Sie richteten ein Wiki[1] ein als zentrales, einfaches Archiv für Projektdokumentation. Sie überwachten die Team-Struktur und änderten gegebenenfalls die Struktur der XP-Paare. Im Rahmen des agilen Ansatzes „fixten“ die Studenten XP zum ersten Mal und passten eine der XP-Praktiken an, in dem sie die Selbstorganisation der Paare beeinflussten.

8 Design von User Stories und collective code ownership

Es gibt andere XP-Praktiken, die angepasst und verbessert werden müssen. In beiden Sessions 2004 und 2005 findet das Erstellen der User Stories kein breites Interesse bei den Team-Mitgliedern. Der Kunde wird als Team-Mitglied wenig zur Kenntnis genommen. Lediglich Projekt- und Qualitätsmanager (in ihrem neuen Rollenverständnis) und einige wenige Team-Members suchen den Kontakt zum Kunden, um maximale Qualität und Verständnis der User Stories zu erreichen. Oftmals werden fertige User Stories „interpretiert“ ohne nachzufragen. Erst wenn Entwickler-Mini-Teams Verantwortung für eine User Story übernehmen, werden sie aktiv und machen ein detailliertes Low-Level-Design.

[1] Ein Wiki (WikiWeb) ist eine im World Wide Web verfügbare Seitensammlung, die von den Benutzern nicht nur gelesen, sondern auch online geändert werden kann. Wikis ähneln damit Content Management Systemen, siehe http://de.wikipedia.org/wiki/Wiki.

Kollektive Verantwortung für den gesamten Code („Collective Code Ownership“) ist im Team nicht sehr beliebt. Außer Projekt- und Qualitätsmanager sind nur noch die wenigen Team-Mitglieder, die schon bei den User-Stories mitgewirkt haben, in der Lage, den gesamten Code zu verstehen – und nicht nur das vergleichsweise kleine Stück, das sie selbst entwickelt haben. Eine Bereitschaft für Collective Code Ownership (XP) ist beiden Sessions nicht erkennbar. Nur die Angst ernsthafte Fehler zu machen, die sich negativ auf das Produkt auswirken, veranlasst wenigstens die oben erwähnten Team-Mitglieder, die genannten XP-Praktiken „auszuprobieren“.

9 Der Weg zur erfolgreichen Software-Integration

Negative Erfahrungen aus der ersten Session des BA-IT-Labors, in der das Team seine Integrationsschritte selbst wählen konnte und dann erst am letzten (!) Tag integrierte, veranlassen den Kunden, in der Session 2005 Integrationszyklen von 3 Wochen zu fordern. Diese werden vom Team aber nicht selbständig eingehalten. Kontinuierliche oder wenigstens konstante Integrationen scheinen für das Team keine Bedeutung zu haben. Jede Minigruppe arbeitet an ihrer eigenen Fragestellung. Massive Intervention des Kunden und ein mehr „klassisches“ Rollenverständnis des Projektmanagers [HBM06] verbessern die Situation. Die Rolle des „Integrations Engineer“ wird geschaffen und mit einem sehr guten, (aber) gerne alleine arbeitenden, Software-Entwickler besetzt. Dieser „Lone Wolf“, der mit seinem Integrations-Rechner immer etwas Abstand zum restlichen Team hat, integriert sehr erfolgreich. Durch die Besetzung dieser Position wird nach ca. 5 Wochen eine kontinuierliche Integration möglich.

Beide Sessions des IT-Labors haben dasselbe Ergebnis: Integration ist nicht das Resultat einer Selbstorganisation des Teams, sondern das Ergebnis einer zentralen Planung des Projektmanagers. Nicht nur der Integrationsrechner muss einzig sein, sondern auch der Posten des Integration Engineer. Integration durch gleichberechtigte Paare, wie in Extreme Programming gefordert, hat im BA-IT-Labor bisher nicht zum Erfolg geführt.

10 Mini-Team-Größe: Sind XP-Paare erfolgreich?

Ein interessanter Parameter für ein Prozessmodell ist die Größe der „Untergruppen“ des Teams (Mini-Teams), in denen die einzelnen User Stories bearbeitet werden. In XP werden „Paare“ gefordert: 2 Entwickler sitzen am selben Rechner, ein Entwickler denkt strategisch, der andere operativ. Um die „Stabilität“ dieser Paare zu analysieren, werden in der Session 2004 zu Beginn Dreierteams gebildet, während in der Session 2005 reine XP-Paare an den Start gehen. Interessant ist die Frage nach der Veränderung der Mini-Teamgrößen über die Zeit (vgl. Tabelle 2).

Session 2004	Session 2005
6 Mini-Teams mit 3 Mitgliedern zum Start.	9 XP-Paare als initiale Mini-Teams.
Im Verlauf des Projekts bilden sich keine XP-Paare spontan.	Die Zahl der XP-Paare nimmt ab.
Die kleinste Teamgröße bleibt 3.	Die meisten Mini-Teams wachsen. Die Zahl der Mitglieder wächst auf 3, 4 und mehr.
Das erfolgreichste Mini-Team hat 4 Mitglieder („Super-Team“ 1).	In speziellen Phasen, z.B. in der Integration, gibt es Teamgrößen mit bis zu 6 Mitgliedern („Super-Team“ 2).

Tabelle 2: Veränderung der Mini-Teamgrößen über die Zeit

Ein paar Beispiele finden sich in den Abbildungen 6 und 7, aufgenommen im Abstand von 2 Wochen:

Abbildung 6: Drei Paare und ein Super-Team im Vordergrund

Abbildung 7: Die Zahl der Paare nimmt ab

Somit scheint es, dass XP-Paare „instabil“ sind. Die Größe der Mini-Teams ändert sich ständig, je nach Aufgabenstellung. Es gibt nach den Erfahrungen der Autoren keinen Grund für die Teamgröße 2, also für Paare. Bei übergreifenden Aufgaben, z. B. bei der Vorbereitung der Integration ist auch eine Sechsergruppe keine Seltenheit. Wenn sich Paare bilden, bestehen sie oft aus einem stärkeren und einem schwächeren Partner, wo der erste die Struktur der Lösung erarbeitet, während der andere die Details erledigt.

Die Organisationspsychologie kennt 6 Verhaltensweisen von Team-Mitgliedern, die alle zu einem erfolgreichen Team gehören müssen: Kommunikatoren, Ideengeber, Fachexperten und Teamworker, die für das Mini-Team wichtig sind, sowie Problemlöser und Qualitätsprüfer, die (zusätzlich) für das gesamte Projektteam von Bedeutung sind (siehe dazu Abbildung 8 sowie [BG01]). Die Mischung aus diesen verschiedenen Verhaltensweisen macht den Erfolg des Teams. „Klassische“ XP-Paare hingegen fordern eher ähnliche Typen, da beide Paar-Mitglieder sich abwechseln sollen im strategischen und taktischen Handeln.

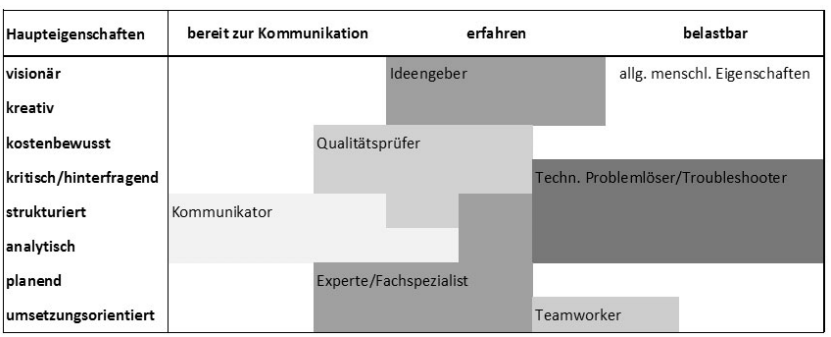


Abbildung 8: Verhaltensweisen im Team

Besonders unter Männern fördert diese Konstellation Rivalitäten, die wiederum den Erfolg des Paars gefährden. Im Rahmen des BA-IT-Labors wurden gut funktionierende, gleichberechtigte XP-Paare nur unter den Studentinnen gefunden!

Mini-Teams, in denen alle oben genannten Projekttypen vertreten sind, und die doch nicht zu groß werden (also vielleicht 4 bis 6 Mitglieder haben), gelten in der Organisationspsychologie als erfolgreich. Dies kann im Rahmen des BA-IT-Labors am Beispiel des Super-Teams aus Abbildung 6 bestätigt werden. Insbesondere dessen erfolgreiches Zusammenführen der Teilaktivitäten im Vorfeld der ersten Integration in der Session 2005 hat den Erfolg dieser Session erst möglich gemacht. Trotzdem ist die Existenz eines Integration Engineers, des „Lone Wolf" unabdingbar, da diese Aufgabe nur zentral bewältigt werden kann.

11 Abweichung des gelebten Prozess vom gewählten Modell

Sowohl die Session 2004 wie auch insbesondere 2005 zeigen, dass der „gelebte" Prozess massiv vom gewählten Prozessmodell abweicht:

2004 gingen die vorgegebenen agilen Dreier-Teams in eine größere Einheit auf. 2005 konnten die XP-Paare nicht „überleben" und wurden ebenfalls durch ein situationsbedingtes größeres Mini-Team „verdrängt" (Abbildung 5 und 6). Außerdem konnte nachgewiesen werden, dass weitere XP-Praktiken, wie das Erstellen von User Stories oder „Collective Code Ownership" in der Praxis nicht gemäß der XP-Regeln, sondern nur stark abgewandelt oder gar nicht umgesetzt wurden. Für die erfolgreiche Software-Integration musste sogar die neue Projektrolle eines „Integrations-Ingenieurs" geschaffen werden.

Diese Erfahrungen wurden in den Sessions 2006 und 2007 bestätigt: 2006 bildete sich statt des vorgegeben XP aufgrund der dominierenden Person des (vom Team gewählten!) Projektmanagers ein streng hierarchischer Prozess mit vielen kleineren Teilprozessen, der aber ebenfalls ein erfolgreiches Produkt erzeugte. Ein Entwicklungsprozess muss eben nicht agil sein, um Erfolg zu haben! 2006 wich der gelebte Prozess also vollständig vom vorgegebenen Prozess ab. Das Team empfand das nicht als Nachteil. Dies ist wieder ein Indiz dafür, dass die Verhaltensweisen der Team-Mitglieder über den (insbesondere von der „Linie") vorgegebenen Prozess dominieren.

12 Crystal Clear – Prozess-Anpassungen durch das Projekt-Team

2007 war ähnlich wie 2005 eine besondere Session: Hier konnte ein Student gewonnen werden, den Prozess des Teams im Rahmen seiner Studienarbeit während des ganzen Semesters zu begleiten. Er wurde zum „Prozessverantwortlichen“ ernannt [Hn07].

Aufgabe des Projekt-Teams war, die bestehende eMail-Marketing Software mit Hilfe des Mono-Projekts [Mp08] auf Linux zu portieren. Als Prozessmodell wurde vom Dozenten „Crystal Clear“ vorgegeben, das agile Prozessmodell für kleinere Teams von Alistair Cockburn [Co04]. Parallel bildete sich bereits beim zweiten Termin eine kleine Untergruppe, die sich mit agiler Dokumentation beschäftigte (Content-Management-System Joomla [Jo08] plus Einsatz eines Wikis der Vorgänger-Teams).

Auf Abbildung 9 und 10 sieht man das zu Projektbeginn gleichmäßig im Raum verteilte Projekt-Team, das bei der ersten größeren Aufgabe (Portierung des bestehenden Systems auf Linux) schon 3 Tage später das erste größere Mini-Team mit 4 Team-Mitgliedern bildet. Damit zeigte die Session 2007 das bereits in den vorhergegangenen Sessions beobachtete Mini-Team-Verhalten: größere Mini-Teams, die sich insbesondere mit „mission-critical“ Aufgaben beschäftigen.

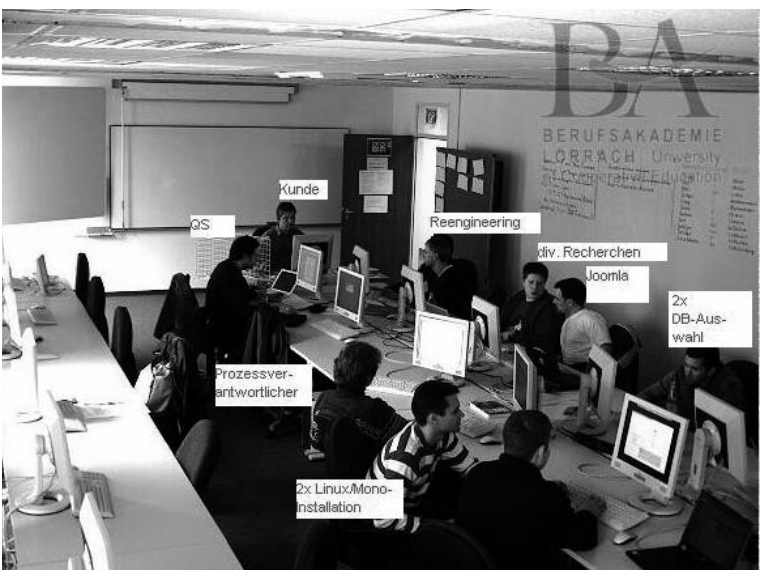


Abbildung 9: Session 2007: Zuordnung der Aufgaben (Projektbeginn)

Abbildung 10: Nur 3 Tage später bildet sich das erste größere Mini-Team (Portierung auf Linux)

2007 wurde das Team zum ersten Mal bezüglich des gelebten Prozess befragt. Crystal Clear fordert im Gegensatz zu XP nur wenige Eigenschaften zwingend:

1. Regelmäßige Lieferung des aktuellen, getesteten Codes,

2. Reflektierte Verbesserung des Prozess, erzielt durch regelmäßig abgehaltene Meetings,

3. Osmotische Kommunikation, d. h. Informationsaustausch im Team findet ganz „nebenbei", ohne zusätzlichen Aufwand (meist durch räumliche Nähe) statt.

Wurden diese Anforderungen an das Team erfüllt? In einer Umfrage sollten die Team-Mitglieder Schulnoten abgegeben bezüglich des Erreichens der oben genannten drei Ziele. Der Prozessverantwortliche analysierte dies im Vorfeld der Umfrage. Insbesondere die unregelmäßigen Code-Lieferungen und die nicht festgestellte Prozess-Reflektion wurden vom Prozessverantwortlichen bemängelt (vgl. Abb. 11).

Wie Crystal-Clear-konform arbeitet das Projektteam?

Projekt	ideal	IT-Labor 2007
Anzahl der Mitarbeiter	• 6 (max. 12)	• 11
Reflexions-Workshop	• < 1 Monat	• nie
Tägliche Standup-Meetings	• ja	• nie
Regelmäßige Lieferung	• Semesterende	• wurde eingehalten
Osmotische Kommunikation	• ja	• ja, ohne Probleme
Iterationslänge	• 3 Wochen	• unregelmäßig
Einfach Kontaktaufnahme mit Endandwendern	• ja	• anfangs nicht oft • während des Projektes immer wieder bei Bedarf
Häufigkeit der Integrationen	• kontinuierlich	• war anfangs nicht möglich, da Portierungsprobleme • gegen Ende wurde häufiger integriert, wenn auch nicht regelmäßig

Abbildung 11: Session 2007: Tabelle aus [Hn07]

Das Team hingegen gab sich selbst hinsichtlich der Erfüllung der Crystal Clear-Anforderungen im Schnitt folgende Schulnoten [Hä07]:

1.	**Crystal-Clear-Eigenschaft**	**Schulnote**
1	Regelmäßige Code-Lieferung	3,0
2	Reflektierte Prozess-Verbesserung	2,9
3	Osmotische Kommunikation	1,7

Diese Umfrage nach ca. drei Viertel der Projektzeit wurde am Ende des Projekts für die Punkte 1 und 2 nochmals durchgeführt und ergab noch bessere Werte:

2.	**Crystal-Clear-Eigenschaft**	**Schulnote**
1	Regelmäßige Code-Lieferung	2,5
2	Reflektierte Prozess-Verbesserung	2,3

Prozessverantwortlicher und Autor konnten lediglich die gute Note für die osmotische Kommunikation erklären. Die anderen Noten waren nicht nachvollziehbar. Deswegen wurden die Team-Mitglieder befragt:

Zum Punkt **(1) Regelmäßige Lieferungen** ergab sich [Hä07]:

„Da es sich bei dem Projekt um eine Portierungsaufgabe handelte, konnte anfangs kaum oder nur sehr schwer fertiger Code geliefert werden. Gegen Ende wurden die Lieferungen dann mehr und auch öfters durchgeführt. Aufgrund der Aufgabe waren regelmäßige Lieferungen nicht möglich“.

Dies ist allerdings nur begrenzt korrekt. Richtig ist wohl, dass das Team keine Notwendigkeit der regelmäßigen Lieferung sah und deshalb das Prozessmodell ungesagt anpasste.

Zum Punkt **(2) Reflektierte Verbesserung** ergab sich [Hä07]:

„Hier wurde vom Team nicht die reflektierte Verbesserung durch ein Meeting verstanden, sondern die vielen Verbesserungen, die durch Gespräche in kleinen Gruppen mit Projektmanager, Qualitätsmanager und auch Kunde erzielt wurden."

Während man bei (1) eventuell noch von einer „Interpretation" der Crystal-Eigenschaft reden könnte, hat das Team bei (2) eindeutig einen eigenen Weg beschritten. Der gelebte Prozess ist nicht mehr Crystal Clear!

13 "Meta Agile Process Model" (MAP)

Im Laufe der (Projekt-) Jahre kommt der Autor mehr und mehr zu der Überzeugung, dass der Ansatz der Entwicklung ständig neuer Prozessmodelle falsch ist. Es ist zwar wichtig, den Studierenden und Auszubildenden fundamentale Prozessmodelle als Projekt-„Baukasten" zur Verfügung zu stellen. Für die Projektpraxis wichtiger aber ist, von diesem „industrialisierten" Ansatz wegzukommen und die minimalen „Zutaten" zu benennen, die ein erfolgreiches Team und damit ein erfolgreicher Prozess benötigt. Man könnte diesen Ansatz als „Meta-Modell" bezeichnen, also als Modell, das beschreibt, welche Elemente ein erfolgreiches Prozessmodell enthalten muss. Der Autor bezeichnet diesen Ansatz als „Meta Agile Process Model" (MAP):

Ein erfolgreiches Projekt in einem nicht zu großen Team (bis vielleicht 20 Mitglieder) braucht nach unseren Erfahrungen aus dem IT-Labor der BA Lörrach folgende Zutaten:

- Ein Kunde, der seine Aufgabe ernst nimmt, und dem Team nicht unbedingt immer, aber genügend oft zur Verfügung steht.
- Ein bis zwei gute und flexible Kommunikations-Manager, die sich Projekt- und Qualitätsmanagementaufgaben aufteilen, aber vor allem die Kommunikation im Team und mit dem Kunden sicherstellen. Sie müssen profunde Kenntnisse in grundsätzlichen „Prototypen" von Prozessmodellen haben und sich nicht scheuen, das angewandte Prozessmodell abzuändern und anzupassen (das ist immer nötig!).
- Ein Integration Engineer, der allein verantwortlich, aber vom Projektmanagement überwacht, eine kontinuierliche Projekt-Integration einfordert und garantiert.
- Ein gut ausgewogenes Team mit einem guten psychologischen Prozess, in dem alle aus der Organisationspsychologie bekannten Typen in ausreichendem Maße vorkommen.

Wenn diese Bedingungen erfüllt sind, spielt nach den Erkenntnissen aus dem BA-IT-Labor das gewählte Software-Entwicklungs-Prozessmodell eine untergeordnete Rolle. Das Team wird im Rahmen der Projekt-Randbedingungen den geeigneten Prozess finden.

Literaturverzeichnis

[Am01] http://www.agilemanifesto.org

[BG01] Baldegger, R.: Erfolgreich im Team. Baldegger Verlag 2004.
Baldegger R.; Gotsmann L.: Ganzheitliches Projektmanagement. Baldegger 2001.

[Ba98] Balzert, H.: Lehrbuch der Software-Technik. Spektrum, 1998.

[Be00] Beck, K.: Extreme Programming Explained. Addison-Wesley 2000.

[Co04] Cockburn, A.: Crystal Clear. Addison-Wesley, 2004.

[Co08] Cockburn, A.: http://alistair.cockburn.us/

[Fo05] Fowler, M.: http://www.martinfowler.com/articles/newMethodology.html#id2251638

[HBM06] Hanser, E.; Brecht, U.; Michelbach, A.: Agile Software-Entwicklung: Extreme Programming im IT-Labor der BA Lörrach. BA Dialog Nr. 6, 2006.

[Ha06] Hanser, E.: Extreme Programming in Real World at the „IT Laboratory“ of BA Loerrach. ISTC Conference Proceedings 2006, Vaal University of Technology, South Africa.

[Hä07] Hättich N.: Das agile Prozessmodell Crystal. Studienarbeit, BA Lörrach, Studiengang Informationstechnik, 2007.

[JBR99] Jacobson I.; Booch G.; Rumbaugh J.: Unified Software Development Process. Addison-Wesley Professional, 1999.

[Jo08] http://www.joomla.de

[Mp08] http://www.mono-project.com

[RBB06] Rausch A.; Broy M.; Bergner K.: Das V-Modell XT, Grundlagen, Methodik und Anwendungen. Springer, 2006.
siehe auch: Dröschel W.; Wiemers M.: Das V-Modell 97. Oldenburg, 2000.

[Sp01] Späth, L.: Die NEW ECONOMY Revolution. Econ 2001.