Seminar in Software Engineering WS 03/04

Agile vs. klassische Methoden der Software-Entwicklung

The Crystal Family

Kapitel 1,3,4,5 ausgearbeitet von:

Silvan Hollenstein
8309 Oberwil

Kapitel 2,6,7 ausgearbeitet von:

Thomas Rutz
5415 Nussbaumen

Institut für Informatik
der Universität Zürich

Dozent: Prof. Dr. Martin Glinz
Assistent: Christian Seybold

Inhaltsverzeichnis

Vorwort

Der erste Teil dieser Arbeit geht auf agile Methoden im Allgemeinen ein, wie sie Alistair Cockburn entwickelt hat. Er hat dabei einiges entdeckt, das als Grundlage für die Crystal Family dient. In unserer Arbeit dient es dazu, etwas über Cockburns grundlegende Gedanken zu erfahren, wieso er schliesslich die Crystal Family entworfen hat.

Cockburn spricht man übrigens als „Co-burn“ aus, mit einem langen o, auf die schottische Art.

Unsere Aufgabe war es ursprünglich, die Themen „Crystal Clear“ und „Crystal Orange“ getrennt zu behandeln. Weil aber die beiden Themen sehr eng beieinander stehen, und sie auf den gleichen Grundprinzipien von Cockburn basieren, entschieden wir uns, eine gemeinsame Arbeit zu schreiben, um so Überlappungen zu vermeiden.

Die Arbeit wurde nun folgendermassen aufgeteilt:

Silvan:

- Einführung und Motivation
- Leichte Methoden nach Cockburn
- Die Crystal Family
- Crystal Clear

Thomas

- Portrait Alistair Cockburn
- Crystal Orange
- Crystal Orange Web

1. Einführung und Motivation

[2] Die Software-Entwickler stehen schon seit langem unter Druck: das Budget kann nicht eingehalten werden, es kann nicht rechtzeitig ausgeliefert werden, oder die Kunden sind unzufrieden, weil das Produkt nicht ihren Vorstellungen entspricht. Immer kürzere Intervalle der Entwicklung und schnell ändernde Kunden-anforderungen verlangen von den Software-Entwicklern, sich dem Markt ständig anzupassen. Hinzu kommt, dass mit der Entwicklung von webbasierter Software ein erhöhter Konkurrenzkampf hinzu gekommen ist. Deshalb ist es besonders wichtig, in diesem Umfeld agil zu bleiben.

Um diese Agilität zu unterstützen, haben sich seit einigen Jahren mehrere Forscher darum bemüht, geeignete Methoden für den Software-Entwicklungs-Prozess zu entwerfen. Ende der 80er-Jahren arbeiteten Kent Beck und Ward Cunningham an einer Methode, die in den späten 90er-Jahren unter dem Namen Extreme Programming bekannt wurde. Mitte der 90er-Jahren konstruierten Ken Schwaber und Jeff Sutherland die Scrum Methode. Die Methode der Crystal Family, die von Alistair Cockburn entworfen wurde, baut auf den gleichen Grundideen auf wie die anderen agilen Methoden.

[2] Gelegentlich sind auch Meinungsverschiedenheiten aufgetreten, die jedoch meistens darauf beruhten, dass die Methoden der Crystal Family nicht richtig verstanden wurden. Das sind meistens Leute, die ganz gerne ein Geheimrezept hätten, dieses einfach stur anwenden können und nicht darauf schauen müssen, ob am Software-Entwicklungs-Prozess im Laufe der Zeit etwas verändert werden sollte. Agile Methoden sind nun mal nicht exakt beschrieben, genau aus dem Grund, weil sie sich ja anpassen müssen. Und deshalb ist die Crystal Family für die einen vielleicht zu schwammig, und für die anderen genau richtig.

Unter all den Methodologie-Forschern hat Cockburn jemanden gefunden, der so viele Gedanken mit ihm teilte, dass sie beschlossen, eine ganze Serie von Büchern über Agile Software-Entwicklung zu schreiben. Dieser jemand war Jim Highsmith, der Begründer von der „Adaptive Software Development“. Alle diese agilen Methoden drehen sich um relativ leichtgewichtige und effektive Software-Entwicklungs-Techniken, die ausserdem den Mensch als soziales Wesen in den Vordergrund stellen.

Die Serie basiert auf zwei Hauptgedanken: [2]

- Verschiedene Projekte benötigen verschiedene Prozesse oder Methoden
- Durch Konzentration auf Fähigkeiten, Kommunikation und Gemeinschaft wird ein Projekt effektiver und agiler als durch Konzentration auf die Prozesse

Den Kernpunkt aller agilen Methoden beschreibt Cockburn folgendermassen:

„Core to agile software development is the use of light-but-sufficient rules of project behavior and the use of human- and communication-oriented rules."

Es geht also darum, eine Methode zu entwickeln, die so leicht wie möglich ist, aber trotzdem alles beinhaltet, was für ein erfolgreiches Projekt nötig ist.

2. Porträt Alistair Cockburn [1]

Die Crystal Methoden verdanken ihre Existenz dem Schotten Alistair Cockburn, der auf eine breite Erfahrung im Bereich der Software Entwicklung zurückgreifen kann.

Alistair Cockburn schloss 1975 in Cleveland, Ohio, ein Studium in Informatik ab. Bis 1984 arbeitete er vor allem an Informatik-Projekten im Bereich der Computer-Grafik. So half er unter anderem bei der Entwicklung eines Flugsimulators mit.

Von 1984 bis 1991 war Cockburn beim IBM Forschungslabor in Zürich beschäftigt. Dort forschte er vor allem im Bereich der Kommunikationstechnologien und unterrichtete während jener Zeit auch Objektorientierte Softwareentwicklung am IBM Education Center in Belgien.

Ab 1991 war Alistair Cockburn in einer beratenden Funktion für IBM tätig. Für den Weltkonzern entwickelte er Methoden zur Objektorientierten Softwareentwicklung. In dieser Zeit analysierte er Dutzende Projekte mit dem Ziel kritische Faktoren im Projektmanagement und der Methodenwahl ausfindig zu machen.

Seit 1994 ist Cockburn in seiner eigenen Firma “Humans and Technology“, die in Salt Lake City zu Hause ist, als Berater tätig. Das Schwergewicht seiner Berater-Tätigkeit liegt nach wie vor im Bereich der Objektorientierten Softwareentwicklung. In dieser Funktion unterstützte er verschiedene Firmen bei der Methodenwahl, der Objektorientieren Softwareentwicklung und dem Projektmanagement. Zu seinen Kunden zählten unter anderem die Central Bank of Norway und eBucks.com (Crystal Orange Web).

Im Februar 2001 unterzeichnet auch Alistair Cockburn das Manifest der Agile Alliance in Utah und wurde so zu einem der 17 Mitbegründer dieser Bewegung. Parallel dazu entwickelte er die “Crystal Methodologies“, seine eigenen Strategien im Bereich der Agilen Softwareentwicklung.

Den Schwerpunkt legt Cockburn bei seinen Studien zweifellos auf den Faktor Mensch. Folgende Aussage gibt er als primäres Ziel an: „Menschliche Faktoren in der Softwareentwicklung verstehen und verbessern.“ In seinen Methoden stellt er einen Punkt immer wieder in den Vordergrund, der bei der Softwareentwicklung gefördert werden soll: Die Kommunikation. Die Verringerung der Kommunikationswege soll den Entwicklungsprozess beschleunigen, bessere Endprodukte liefern und zufriedene Mitarbeiter schaffen. Die Wahl der technischen Hilfsmittel lässt Cockburn aber bewusst offen, diese soll von den Projektpersonen selbst gefällt werden.

3. Leichte Methoden nach Cockburn

Im Englischen gibt es den Begriff „Methodology“, welcher aber laut Duden zwei leicht unterschiedliche Bedeutungen hat: Zum einen ist es eine Serie von zusammengehörigen Methoden oder Techniken, zum anderen die Lehre der Methoden. Für die erstere Bedeutung gibt es auf deutsch keine entsprechende Übersetzung. Weil diese Arbeit aber genau dieses Thema behandelt, wird im folgenden einfach das Wort Methode verwendet.

[2] Jede Organisation hat eine Methode, zwar nicht auf den ersten Blick offensichtlich, doch geht es hauptsächlich darum, wie sie ihre Tätigkeiten, ihr Geschäft ausführt. Eine Methode für die Software-Entwicklung beinhaltet zum Beispiel: „welche Leute stellt man ein, wie arbeiten sie miteinander, welche Hilfsmittel benutzen sie, wie kommunizieren sie miteinander, wie teilen sie ihr Wissen usw.“ Sie beschreibt aber auch Konventionen, oder die Kultur im Team, das soziale Verhalten untereinander. Cockburn verwendet auch den Begriff Ökosystem einer Unternehmung.

3.1. Software-Entwicklung als kooperatives Spiel

Was ist Software-Entwicklung eigentlich? Ist es Kunst, Handwerk, Wissenschaft, Technik, oder etwas ganz anderes? Auch wenn es für einen beherzten Programmierer sicherlich ein Kunstwerk ist, geht es doch in erster Linie darum, lauffähige, fehlerfreie Software auszuliefern. Leider ist aber Software-Entwicklung kein Prozess, den man ohne weiteres steuern kann. Vielmehr wird die Software neu erfunden, oder zumindest aus bestehenden Teilen zusammengesetzt oder ergänzt. Sicher ist nur, dass es keinen klar definierten Prozessablauf gibt, der garantieren würde, dass die Software in einer bestimmten Zeit fertiggestellt ist.

[2] Cockburn umschreibt die Software-Herstellung *als „A cooperative Game of Invention and Communication“*. Er vergleicht sie mit Bergsteigen in einer Gruppe. Auch dort ist Kooperation wichtig, es ist zielgerichtet und hat irgendwann ein Ende.

Bei der Software-Entwicklung gibt es nichts anderes als Ideen (Erfindungen) in den Köpfen der Leute, und Kommunikation. Dabei können beide auf Hilfsmittel zurückgreifen.

Ideen basieren auf Gedankenkonstrukten. Der Mensch kann ein solches Konstrukt nicht endlos aufbauen, sondern muss irgendwann die bisherigen Gedanken aufschreiben. Er zeichnet sich eventuell Modelle, und kann damit auch wieder neue Ideen generieren.

Die Kommunikation kann folgendermassen stattfinden:

- direkter Informationsaustausch im Team
- Mitarbeiter erzeugen Dokumente, Videos, oder was immer angebracht ist, um nachfolgenden Generationen das Wissen weiterzuvermitteln
- Mitarbeiter erzeugen Notizen, um sich selber wieder zu erinnern

[2] All diese Hilfsmittel sind gut, solange sie dem eigentlichen Ziel, das Spiel zu gewinnen, nicht im Wege stehen. Und das Spiel gewinnen bedeutet, die Software wird rechtzeitig und defektfrei geliefert. Das zweite Ziel ist es, eine gute Ausgangslage für das nächste Spiel zu haben, bei welchem vielleicht das bestehende System geändert oder ersetzt wird, oder ein ähnliches System entwickelt wird. Die Hilfsmittel und Zwischenprodukte dienen nur dazu, diese Ziele schneller zu erreichen.

Die Frage stellt sich nun, wie viele Zwischenprodukte nötig sind, um diese Ziele zu erreichen. Und genau hier setzen leichtgewichtige Methoden wie die Crystal Family an. Es soll exakt so viel hergestellt werden wie nötig ist, aber ja nicht mehr. Da ein Software-Entwicklungs-Projekt limitiert ist an Zeit und Ressourcen, ist ein ständiges Abwägen wichtig, wie viel Zeit in Zwischenprodukte investiert werden soll.

Nur das erste Ziel vor Augen, hält sich das Problem noch in Grenzen. Es werden so viele Dokumente geschrieben wie nötig sind, um das eigene Gedächtnis wieder aufzufrischen, Gedankenkonstrukte festzuhalten, oder um Wissen innerhalb des Teams auszutauschen (falls dies nicht in direktem Kontakt geht). Beachtet werden sollte noch die Gefahr, Wissen zu verlieren, wenn Mitarbeiter aus der Firma ausscheiden.

Kommt das zweite Ziel hinzu, ist die Problematik einiges komplexer. Jetzt muss zusätzlich beachtet werden, dass genügend Dokumente angefertigt wurden, um beim nächsten Spiel wieder leicht ansetzen zu können. Oder in der Sprache eines Bergsteigers ausgedrückt: man möchte in den nächsten Jahren noch viele andere Berge hinaufklettern. In einer Software-Unternehmung wird das Team einer ständigen Wandlung unterworfen sein, es ist deshalb entscheidend, dass altes Wissen erhalten bleibt.

3.2. Der Mensch als Individuum

[2] Dass Menschen Fehler begehen, ist nichts Neues. Doch Cockburn hat immer wieder missglückte Projekte gesehen, bei denen diese menschliche Charakteristik nicht in Betracht gezogen wurde. Es waren dies fünf Arten von menschlichen Schwächen, die offenbar zu einem Misserfolg des Projekts führten, wenn sie zuwenig beachtet wurden.

Die fünf Unzulänglichkeiten sind:

- Menschen begehen Fehler
- sie scheitern lieber konservativ
- sie erfinden lieber neu, statt nachzuforschen
- sie lieben Gewohnheiten
- sie sind inkonsistent

Aus dem Umstand, dass Menschen Fehler begehen, ist man von der seriellen zur iterativen und inkrementellen Entwicklung übergegangen. Denn dies hat den Vorteil, dass man im Laufe der Entwicklung dazu lernen kann, einzelne kleine Teile des Gesamtsystems besser überblickbar sind und nach jedem Intervall eine Anpassung möglich ist.

Menschen haben Angst davor, Neues auszuprobieren. Wenn die herkömmliche Vorgehensweise einigermassen gut funktioniert, wird oftmals das Risiko gescheut, zu einer neuen Vorgehensweise überzugehen. Denn das Risiko besteht, dass es auf die neue Art gut funktioniert, oder eben das Gegenteil.

Menschen haben die Tendenz, Dinge neu zu erfinden, als schon bestehendes zu übernehmen oder abzuändern. Das kommt in der Software-Entwicklung natürlich besonders deutlich zum Tragen, da dort die Wiederverwendung zu *dem* Schlagwort geworden ist. Der Mensch neigt aber dazu, neuen Code zu schreiben. Dies ist auch bekannt unter dem Begriff „Not-Invented-Here“.

Menschen lieben Gewohnheiten. Deshalb ist es schwierig, die Leute zu neuen Software-Entwicklungsmethoden zu bewegen. Sogar wenn man den optimalen Entwicklungsprozess gefunden hat, die Leute würden sich am Anfang sträuben, ihn zu verwenden, und wenn doch, würden sie ihn sporadisch oder nachlässig verwenden. Erst wenn die Gewöhnung an das Neue wieder einsetzt, halten sie sich an die Arbeitsweise der Methode.

Nicht nur, dass der Mensch Mühe hat, neue Handlungsweisen anzunehmen, er ist darüber hinaus auch noch inkonsistent in seinem Handeln. Das heisst, dass seine Arbeitsweise von Stunde zu Stunde ändern kann, und deshalb nicht genau einschätzbar ist für einen Projektleiter. Für diesen ist wohl die schwierigste Aufgabe bei der Einführung einer neuen Methode, die Leute dazu zu bewegen, neue Gewohnheiten anzunehmen und gleichzeitig konsistent zu sein in dieser Veränderung.

[2] Bei der Erfindung einer Methode hat man die Wahl zwischen Disziplin und Toleranz. Bei den klassischen Methoden hat man meistens Disziplin gewählt, obwohl das im Gegensatz zum inkonsistenten Verhalten steht.

Führt man eine hochdisziplinierte Methode ein, ist es sehr wahrscheinlich, dass die Leute nach kurzer Zeit sich nicht mehr an alle Regeln halten. Ein Beispiel dafür ist Personal Software Process (PSP):

Beim PSP muss jeder Programmierer aufschreiben, wie lange er für was gebraucht hat und wo Fehler entstanden sind. Aufgrund dieser Notizen kann diese Person das nächste Mal seine Konzentration mehr auf die fehleranfälligen Arbeiten lenken. Das Schwierige daran ist aber, dass es einen Mehraufwand bedeutet, diese Notizen zu machen, dass es Disziplin erfordert. PSP hat keine speziellen Mechanismen, um diese hochdisziplinierten Arbeiten am Laufen zu halten. Es würde deshalb nicht überraschen, wenn die Arbeiter schon nach kurzer Zeit diese Notizen nicht mehr schreiben würden.

Cockburn betont aber, dass hochdisziplinierte Methoden nicht unbrauchbar sind, sondern sie sind einfach fragil.

Auch tolerante Methoden sind nicht ohne Probleme. Bei ihnen muss jeder Mitarbeiter Verantwortung übernehmen, weil weniger Prozesse vorhanden sind und damit auch weniger Kontrolle über jeden einzelnen.

3.3. Entwurf einer Methode

[2] Wie werden neue Methoden entwickelt? Cockburn stützt sich dabei vor allem auf die eigenen Erfahrungen und die Erfahrungen von anderen Projektleitern, die er interviewte. Explizit geht er so vor:

- I adjust, tune, and invent whatever is needed to take the project to success.
- After the project, I extract those similar circumstances and add them to my repertoire of tactics and strategies.
- I listen to other project teams when they describe their experiences and the lessons they learned.

3.3.1. Entwurfsfehler

[2] Cockburn hat festgestellt, dass Neulinge auf dem Gebiet der Methodenentwicklung oft die gleichen Fehler begehen:

Eine Grösse für alle Projekte: Ob eine Methode gross oder klein, bzw. schwer oder leichtgewichtig sein soll, hängt immer von jedem einzelnen Projekt ab. Die Einheitsgrösse gibt es nicht.

Intolerant: Die Leute verhalten sich nicht alle gleich, und haben verschiedene Ansichten und Arbeitsweisen. Eine Methode muss deshalb genügend Spielraum haben, um alle Charakteristiken einbeziehen zu können.

Zu schwer: Leute, die nach einer traditionellen Methode vorgehen, hegen die Annahme, dass ein Projekt sicherer ist mit rigoroser Überwachung und vielen Dokumenten, also mit einer schwergewichtigen Methode. Dies kommt wahrscheinlich daher, dass der Projektleiter im allgemeinen keinen Code sieht. Dadurch entsteht eine gewisse Angst, und es resultiert das Verlangen nach mehr Dokumentation. Die Alternative wäre Vertrauen. Einer Methode mehr Gewicht hinzuzufügen als nötig ist, bedeutet aber, dass die Wahrscheinlichkeit, die Software rechtzeitig auszuliefern, abnimmt, und nicht zunimmt. Denn mit mehr Gewicht sind die Leute nur noch mit dem Schreiben von Dokumenten beschäftigt. Die Schwierigkeit besteht darin, zu realisieren, wann man Vertrauen schenken kann und wann nicht. Und welche Einschränkungen sind mehr eine Last als Sicherheit.

Unversucht: Bei all den verschiedenen Methoden, die Cockburn schon beobachtet hat, ist ihm aufgefallen, dass beim Entwerfen einer Methode nichts offensichtlich ist. Viele Dinge, die den Anschein haben, dass sie funktionieren, klappen eben nicht und umgekehrt. Das paarweise Programmieren und Test-first-Development zum Beispiel sehen nicht so aus, als würden sie gut funktionieren, tun es aber doch, wie die Erfahrung gezeigt hat.

Einmal verwendet: Eine Methode, die einmal erfolgreich angewendet wurde, kann noch nicht als generelle Lösung angesehen werden. Tatsache ist, dass für jedes Projekt eine massgeschneiderte Methode gebraucht wird. Die Kunst im Entwickeln einer Methode besteht eben auch darin, das Gebiet ihrer Anwendbarkeit festzulegen.

3.3.2. Erfolgreiche Projekte

[2] Es stellt sich die Frage, wie man ein Projekt beurteilen kann. Cockburn hat dafür drei Kriterien aufgestellt:

- The project was delivered. I don't ask if it was completed on time and on budget, just that the software went out the door and was used.
- The leadership remained intact. They didn't get fired for what they were doing.
- The people on the project would work the same way again.

Beim ersten Kriterium hält er die Messlatte bewusst tief, weil jeder den Erfolg eines Projektes wieder anders beurteilen würde. Das zweite hat er eingeführt, nachdem er ein laufendes Projekt besucht hatte, das als erfolgreich eingestuft wurde. Wenig später wurde der Projektleiter gefeuert, weil über ein Jahr kein Code produziert worden war. Das dritte Kriterium mag seltsam sein, doch für Methoden, bei denen

der Mensch im Mittelpunkt steht, sollte auch für das Wohlbefinden und die Arbeitsmoral der Leute gesorgt sein, denn wer Spass an der Arbeit hat, ist bekanntlich auch motivierter. Cockburn hat beim dritten Punkt noch eine weitere Hürde gestellt: nur wenn die Leute wirklich von sich aus die Methode ein weiteres Mal verwenden würden, ist der Punkt erfüllt, d.h. sie verwenden sie, auch wenn der Aufpasser nicht dabei ist.

Bisher hat Cockburn drei Methoden entdeckt, bei denen die Leute gewillt waren, sie ein weiteres Mal zu verwenden. Dies waren:

- Responsibility-Driven Design (Wirfs-Brock 1990)
- Extreme Programming (Beck 1999)
- Crystal Clear (Cockburn) [4]

Crystal Orange hat er nicht auf die Liste gesetzt, weil er selber das Projekt beraten hat und Prozess Designer war.

3.3.3. 7 Prinzipien

[2] Cockburn hat sieben Prinzipien herausgefunden, die nützlich sind, wenn man eine Methode entwickelt.

Prinzip 1: Interaktive, persönliche Kommunikation ist die billigste und schnellste Art des Informationsaustausches

Dies besagt, dass Leute mit direktem Kontakt es einfacher haben, Software zu entwickeln, und die Software wird billiger sein. Wenn die Projektgrösse zunimmt, ist natürlich der direkte Kontakt nicht mehr zu allen möglich, ausserdem nimmt die Anzahl der Kommunikationswege überproportional mit der Anzahl Leute zu. Mit der Projektgrösse nehmen also auch die Kosten der Kommunikation zu und die Schwierigkeit der Software-Entwicklung steigt.

Anders ausgedrückt, die Effektivität einer Person nimmt ab, je mehr Leute beteiligt sind. Wenn also die Produktivität und Kosten im Vordergrund stehen und der Liefertermin egal ist, ist es vorteilhafter, kleine Teams zu haben.

Prinzip 2: Übergewichtige Methoden sind teuer

Es soll nur das wirklich nötigste produziert werden, und was darüber hinaus geht, ist mit Kosten verbunden. Wenn die Programmierer neben der wichtigsten Arbeit, dem Code schreiben, noch viel bürokratisches Material erzeugen müssen, werden sie zwangsläufig im Arbeitsfluss gebremst. Kleine Teams kommen mit wenigen Dokumenten zurecht, verwenden also eine leichtgewichtige Methode. Je grösser aber ein Projekt ist, desto mehr Dokumentationen müssen geschrieben werden,

umso schwerer wird die verwendete Methode. Deshalb ist es einleuchtend, dass kleinere Teams effizienter arbeiten können.

Die Methode kann aber auch zu leichtgewichtig werden. Dann kann die Qualität des Codes darunter leiden, oder die Programmierer sind nur noch damit beschäftigt, alten Code zu korrigieren.

Prinzip 3: Grössere Teams brauchen schwerere Methoden

Ein Team von bis zu zehn Leuten ist noch gut überschaubar. Werden es aber 30 Leute oder mehr, sind schon besondere Massnahmen der Koordination nötig, man weiss nicht mehr genau woran die anderen gerade arbeiten, es muss darauf geachtet werden, dass sich die Arbeiten nicht überlappen oder in die Quere kommen. Kommunikation und Koordination sind also umso wichtiger, je grösser das Team ist.

Prinzip 4: Je kritischer ein Projekt, desto schwerer die Methode

Die Risiken, die bei einer Fehlfunktion der Software eingegangen werden, unterteilt Cockburn in vier Kategorien:

- Loss of comfort
- Loss of discretionary monies
- Loss of essential monies
- Loss of life

Mit “discretionary monies” meint Cockburn Einnahmen oder Überweisungen mit einem falschen Betrag, die aber durch eine Absicherungsprozedur bemerkt und richtiggestellt werden können. Banktransaktionen gehören auch in diese Kategorie, auch wenn es verwunderlich tönt.

Mit „Essential monies“ ist z.B. gemeint, dass durch einen Softwarefehler eine Unternehmung bankrott gehen könnte.

Prinzip 5: Je mehr Feedback und Kommunikation, desto weniger Dokumentationen sind nötig

Durch inkrementelle Entwicklung kann man laufend kleine, lauffähige Teile der Gesamtsoftware liefern. Der Auftraggeber kann sich darauf mit den Spezialisten für Anforderungsspezifikationen zusammensetzen und Korrekturen vornehmen. Dadurch entfallen ständige Anforderungs-Reviews innerhalb des Teams. Die Auswirkungen einer vorausgegangenen Tätigkeit können schneller erkannt werden, und dadurch verringern sich ständige Nachkorrekturen.

Prinzip 6: Disziplin, Fähigkeiten und Wissen versus Prozesse, Formalismus und Dokumentation

Jim Highsmith [6] hat es so ausgedrückt: *Verwechsle nicht Dokumentation mit Wissen.* Was er meint, ist, dass sehr viel mit einem Projekt verbundenen Wissen nur in den Köpfen der Leute existiert und nicht auf Papier. Auch mit der besten Dokumentation kann ein neues Team nicht dort anknüpfen wo das alte aufgehört hat.

Highsmith weiter: *Prozess ist nicht Disziplin*. Disziplin bedeutet, eine Person wählt von sich aus eine bestimmte Art zu arbeiten, Prozess ist, die Person zu einer bestimmte Arbeitsweise zu zwingen. Natürlich hat das erstere davon einen besseren Einfluss auf ein Projekt.

Die dritte Unterscheidung von Highmith ist: *Verwechsle nicht Formalismus mit Fähigkeiten*. Gemeint ist, dass eine Person mit überragenden Fähigkeiten nicht ersetzt werden kann durch eine Person mit mittelmässigen Fähigkeiten, die ihre Arbeit mittels Formalitäten und Anleitungen erledigt.

Prinzip 7: Effizienz ist erweiterbar in Aktivitäten, die nicht den Flaschenhals eines Prozesses darstellen

Angenommen, in einem Projekt sind fünf Leute mit dem Programmieren von Smalltalk beschäftigt, aber nur einer ist für die Programmierung der Datenbank zuständig, und die Datenbankerstellung sei der nachgelagerte Prozess. Dann gibt es zwei Möglichkeiten: Entweder schickt man vier Smalltalk-Programmierer nach Hause, oder aber, wenn das Projekt möglichst schnell abgeschlossen sein soll, setzt man die fünf Programmierer effizient ein. Das bedeutet, dass die fünf Leute ihre Arbeit absolut gründlich und fehlerfrei erledigen, dass wenn sie ihr Design dem Datenbank-Programmierer übergeben, dieser es mühelos lesen kann und seine Arbeit somit in kürzerer Zeit abgeschlossen sein kann.

3.4. Agil und anpassungsfähig

[2] Cockburn gibt selber eine Definition für agile Methoden ab: *Agile implies being effective and maneuverable. An agile process is both light and sufficient. The lightness is a means of staying maneuverable. The sufficiency is a matter of staying in the game. The question for using agile methodologies is not, "Can an agile methodology be used in this situation?" but "How can we remain agile in this situation?"*

Es ist wohl klar, dass ein 40-Personen Team nicht so agil sein kann wie ein 10-Personen Team. Jedoch kann ein Team unabhängig von seiner Grösse seine Agilität maximieren. Es geht darum, regelmässige Besprechungen abzuhalten, wo über die

angewandte Methode diskutiert wird. Agil heisst ja, dass die Methode mit der Zeit, wenn es die Umstände verlangen, verändert und der Umgebung angepasst wird.

Das Ziel ist es, wie Cockburn es ausdrückt, sogenannte Sweet Spots auszumachen und anzusteuern. Ein Sweet Spot ist eine Situation, wo die Software effektiv entwickelt werden kann.

[2] Die fünf Sweet Spots sind:

Zwei bis acht Leute in einem Raum

Wie bereits erwähnt, hat man bei bis zu acht Leuten den Vorteil, die billigste und effektivste Art der Kommunikation zu führen. Ausserdem braucht es bei dieser Grösse noch keine verwaltungsbezogene Arbeiten wie Sekretariat.

Anwendungsexperte Vor-Ort

Damit ist sofortiges Feedback von einem zukünftigen Anwender garantiert. Neue Ideen der Entwickler können rasch umgesetzt werden und es besteht nie die Gefahr, in die falsche Richtung zu gehen.

Einmonatige Inkremente

Die beste Möglichkeit Feedbacks zu erhalten, besteht in der inkrementellen Entwicklung. Es wird unvermeidlich sein, bei gewissen Projekten die Inkremente auf mehrere Monate zu verlängern. Cockburn hat aber in Interviews erfahren, dass die Leute üblicherweise kürzere Inkremente bevorzugen, und ihre Konzentration auf ein Projekt nach etwa drei Monaten nachlassen.

Vollautomatische Regressionstests

Unit- und Functional Tests haben zwei Vorteile: Die Entwickler können ihren Code überarbeiten und mit einem Knopfdruck gleich wieder testen. Durch die Tests können sich keine kleinen Fehler einschleichen. Ausserdem hätten Leute berichtet, dass sie mit diesen Tests das Wochenende besser geniessen können, weil sie, wenn sie am Montag das Büro betreten, sofort sehen können, wenn ihr Code verändert wurde.

Erfahrene Entwickler

In einer idealen Situation besteht das Team nur aus erfahrenen Entwicklern, weil diese etwa zwei bis zehn mal effektiver sein können als unerfahrene Entwickler. Die Anzahl der Entwickler kann somit massiv verringert werden, wenn das Team vollständig nur aus erfahrenen Leuten besteht.

4. Die Crystal Family [2]

Jedes Projekt benötigt immer eine auf sich selber zugeschnittene Methode. Daher kann ein Methodenentwickler einem Team nicht einfach eine Methode als allgemeingültig vorstellen, sondern hat dabei zwei Vorgehensweisen:

- Er kreiert ein Set von Methoden-Einzelteilen, die dann ein Team für ihr Projekt zusammensetzt.
- Er kreiert eine Familie von Methoden, das Team wählt die passendste für sich aus und ändert diejenigen Teile, die nicht auf das Projekt passen

Der Rational Unified Process ist ein solches Set von Methoden-Einzelteilen, ein Framework um die geeignete Methode zu konstruieren. Die Crystal Family hingegen besteht aus konkreten, erfolgreich verwendeten Methoden. Diese dienen als Basis, um die eigene Methode zu schneidern.

Die Unterschiede innerhalb der Crystal Family kommen durch unterschiedliche Projektgrössen und Risiken zustande.

Wie auf der folgenden Grafik ersichtlich, ist auf der waagerechten Achse die Projektgrösse und auf der senkrechten das mit der Software verbundene Risiko. C steht für „Loss of Comfort“, D für „Loss of discretionary monies“, E für „Loss of essential monies“ und L für „Loss of Life“. Die Zahl dahinter gibt die Anzahl der an einem Projekt beteiligten Leute an. Zum Beispiel wäre E40 ein Projekt mit essentiellen Geldern und 40 Leuten.

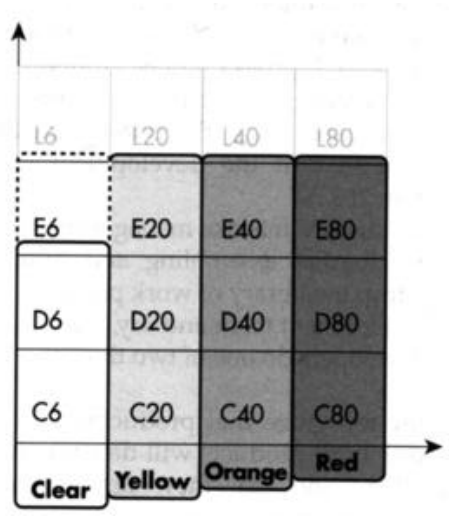

Abbildung 1: Crystal Family, sortiert nach Farben

Projekte mit lebenskritischen Systemen hat Cockburn bewusst weggelassen, weil er selber noch nie in solchen Projekten tätig war oder Leute interviewt hätte, die das gemacht haben.

Das Feld E6 ausserhalb von Crystal Clear bedeutet, dass Clear nicht unbedingt Projekte mit essentiellen Geldern anspricht. Das Team könnte aber trotzdem Crystal Clear als Basis verwenden und auf die Situation ausdehnen.

Eine andere Einschränkung der Crystal Methoden ist, dass sie nur eng miteinander verbundene Teams ansprechen. Räumlich verteilte Teams haben nicht die nötigen Voraussetzungen für das Anwenden von Crystal, ausser sie treffen Massnahmen, um die räumliche Distanz zu überbrücken, z.B. Videokonferenzen, Chat, usw.

Crystal erhebt auch keinen Anspruch darauf, auf- oder abwärtskompatibel zu sein. Mit anderen Worten, wenn ein Projekt mit vier Personen auf 20 anwächst, kann nicht gefragt werden: „können die alten Arbeitskonventionen beibehalten werden?“, sondern: „Wie können 20 Leute gut miteinander arbeiten?“

Die Hauptelemente von Crystal

Die Hauptphilosophie von Crystal ist, dass Software-Entwicklung als ein kooperatives Spiel von Erfindung und Kommunikation angesehen wird, mit dem Hauptziel, nützliche und funktionierende Software auszuliefern, und dem zweiten Ziel, dafür zu sorgen, dass auch zukünftige Projekte erfolgreich sind.

Crystal legt ausserdem sehr viel Wert darauf, das Können der einzelnen Leute und des ganzen Teams ständig zu verbessern.

Mitglieder der Crystal Family haben gemeinsam:

- Werte und Prinzipien
- On-the-fly tuning (ständige Anpassungen)

Die zwei Hauptwerte der Crystal Methoden sind:

- Mensch- und Kommunikationsorientiert
- Hochtolerant

Das erste besagt, dass Werkzeuge, Zwischenprodukte und Prozesse nur der Unterstützung des Menschen dienen. Oder wie man im *Agile Software Development Manifesto* zur Übereinstimmung gelangt ist: *Individuals and interactions over processes and tools*. Das zweite spricht auf die unterschiedlichen Kulturen an.

Weil aber Crystal relativ viel Spielraum gibt, kann das Team trotzdem wählen, in einer hochzeremoniellen oder hochdisziplinierten Art zu arbeiten, z.B. indem sie Teile von PSP oder XP übernimmt.

Ein sehr wichtiges Prinzip in Crystal ist die Kommunikation mit dem Anwender. Zur Veranschaulichung ein Beispiel:

Der Kunde weiss nicht genau, was er will. Die Programmierer bestehen aber darauf, dass der Kunde ihnen sagen soll, was zu tun ist. Also sagen die Kunden dann irgendwas, worauf die Programmierer genau das entwickeln.

In diesem Fall wäre es besser, die Programmierer helfen den Anwendern zuerst einmal, Erfahrungen zu sammeln, wie sie sich ausdrücken können. Dies kann zum Beispiel auch ein Schnellkurs sein für Use Cases. Die Programmierer würden die gezeichneten Use Cases vorlegen und die Anwender könnten dann entscheiden, ob

das ihren Wünschen entspricht. Die zweite Möglichkeit wäre, dass die Programmierer zuerst einen Prototype entwerfen, um festzustellen, ob das dem Kunden gefällt.

Das ist genau was Crystal will: Ständige Kommunikation mit dem Anwender, auch während der Entwicklungsphase. Denn so arbeiten Leute aus verschiedenen Gebieten zusammen und können sich optimal ergänzen.

Die ganze Crystal Family hat zwei Regeln gemeinsam:

- Das Projekt bedient sich der inkrementellen Entwicklung, mit Inkrementen von vier Monaten oder weniger (vorzugsweise ein- bis dreimonatige Inkremente)
- Das Team muss vor und nach einem Inkrement Besprechungen abhalten, wenn möglich, auch in der Mitte eines Inkrements.

Die zwei Haupttechniken in Crystal sind:

- Die Technik der Methodenanpassung: Am Anfang eines Projekts werden vergleichbare Projekte herangezogen um damit die passendste Methode zu eruieren. Im Team wird dann ein Workshop abgehalten, wo diese Basismethode dem eigenen Projekt angepasst wird.
- Die Technik, Reflection Workshops abzuhalten

Cockburn betont aber, dass diese zwei Techniken auch ersetzt werden können, solange die Ziele erreicht werden.

Im folgenden werden die drei Crystal Methoden beschrieben, die Cockburn schon im Einsatz erlebt hat.

5. Crystal Clear

5.1. Anwendungsbereich

[4] Bei Crystal Clear gibt es nur ein Team, das sich entweder im gleichen Raum oder in benachbarten Räumen befindet.

Es ist eine Methode für Projekte der Kategorie C6 und D6 (siehe Abbildung 2). Es sollte aber möglich sein, Crystal Clear auf ein Projekt der Kategorie E8 oder D10 auszudehnen, es muss dann aber besonders auf die Kommunikation und das Testen geachtet werden. Die Methode noch mehr zu erweitern erachtet Cockburn aber als schwierig, weil Crystal Clear keine Kommunikationsstrukturen beinhaltet für Teams, die nicht im selben Raum arbeiten können. Ausserdem kann Crystal Clear nicht für lebenskritische Systeme angewendet werden, weil die Systemvalidierung hierfür fehlt.

5.2. Rollen in Crystal Clear

[4] Es gibt Aufgaben, die notwendigerweise von verschiedenen Leuten erfüllt werden müssen:

- Auftraggeber
- Senior Designer-Programmierer
- Designer-Programmierer
- Anwender (Minimum Teilzeit)

Jemand von diesen Personen kann die Aufgabe des Projektleiters übernehmen. Eine Person ist der Geschäftsexperte, und entweder einer oder mehrere Leute sind für die Anforderungsspezifikation zuständig.

Der Senior Designer-Programmierer ist die Schlüsselperson im Team. Die anderen Designer-Programmierer müssen keine Profis sein, es genügt, wenn es eine Mischung aus Anfängern und Fortgeschrittenen ist. Es hängt aber im Einzelfall davon ab, inwieweit der Senior Programmierer die anderen unterstützen kann, und wie schwierig das Projekt ist. Gut ausgebildete Leute einzustellen ist aber sicher von Vorteil.

5.3. Praktiken

[4] Die Praktiken in Crystal Clear sind:

- Software wird inkrementell und regelmässig ausgeliefert, alle zwei bis drei Monate
- Der Fortschritt wird durch Meilensteine verfolgt. Die Meilensteine bestehen aus Softwarelieferungen oder wichtigen Entscheidungen, im Gegensatz zu geschriebenen Dokumenten
- Es gibt gewisse automatisierte Regressionstests
- Die Anwender sind in die Entwicklung involviert
- Pro Release werden zwei User Viewings durchgeführt
- Nachgelagerte Aktivitäten beginnen, sobald Vorgelagerte stabil genug für ein Review sind.
- Am Anfang und in der Mitte eines Inkrements werden Workshops abgehalten, wo allfällige Änderungen am Produkt und die angewendete Methode besprochen werden.

Diese Praktiken müssen so eingehalten werden, könnten aber auch durch Praktiken von Scrum, XP oder Adaptive Software Development ersetzt werden. Zum Beispiel könnte von Scrum die täglichen Meetings, oder von XP das Pair Programming übernommen werden.

5.4. Arbeitserzeugnisse und Tools

[4] Die Arbeitserzeugnisse sind:

- Release Folgen
- Zeitplan mit den User Viewings und Arbeitserzeugnissen
- kommentierte Use Cases oder Anwendergeschichten
- Design Entwurf und, falls benötigt, andere Notizen
- Bildschirm Entwürfe
- gemeinsames Objektmodell
- lauffähiger Code
- Testfälle
- Benutzerhandbuch

Bei folgenden Erzeugnissen wird es dem Team überlassen, ob es sie anfertigen möchte oder nicht:

- Vorlagen für Dokumente

- Standards für den Code und die Benutzerschnittstelle
- Standards und Details der Regressionstests

Crystal Clear verlangt Projektdokumentation; nur was sie alles beinhaltet, ist nicht vorgegeben, sondern sollte im Einzelfall entschieden werden. Das ganze Team muss beurteilen können, wie viel Dokumentation des Systems nötig ist, damit zukünftige Teams leichter anknüpfen können.

Die wichtigsten Werkzeuge, die ein Team besitzen kann, sind:

- Ein Versionen- und Configuration-Management System
- Eine Whiteboard Tafel, mit der ausdrucken möglich ist

Ein Whiteboard zum Ausdrucken ist deshalb nützlich, weil nach einer Besprechung jeder die Notizen automatisch zur Verfügung hat und niemand zuerst eine Zusammenfassung der Besprechung eintippen muss.

5.5. Vergleich zu XP

[5] Crystal Clear hat sehr viele Gemeinsamkeiten mit Extreme Programming. In beiden Methoden wird sehr viel Wert auf das Feedback gelegt. Deshalb ist der Kunde auch in beiden Methoden in den Entwicklungsprozess integriert, und wenn möglich steht er den Programmierern rund um die Uhr zur Verfügung. XP hat aber zusätzlich als Feedback die automatischen Tests zur Verfügung, was bei Crystal Clear nicht der Fall ist. Auch dort hat man zwar zum Teil automatische Tests, aber die werden viel lascher gehandhabt, das heisst, es wird nicht nach jeder kleinen Änderung wieder ein Test laufen gelassen. Überhaupt sind in XP nur sehr kurze Pläne erlaubt, alles was über einen Tag hinaus geht, ist schon zu weitsichtig. Dies wird in Crystal Clear nicht so praktiziert, hier wird in der Regel für ein Inkrement geplant, was etwa ein bis drei Monate sind.

Was auch beiden gemeinsam ist, ist der Fokus auf den Menschen im Team, und die inkrementelle Entwicklung.

Das einzige, worin sich XP und Crystal Clear wirklich unterscheiden, ist die Handhabung mit der Dokumentation. In XP wird für hier und jetzt gedacht, Dokumentation ist nicht wirklich wichtig, weil das Wissen in den Köpfen des Teams steckt. Cockburn ist in diesem Punkt anderer Meinung. Er findet Dokumentation bis zu einem gewissen Grad nötig, denn wer weiss, was in Zukunft mit dem Team passiert. Was ist, wenn die Hälfte der Leute aus dem Team aussteigen, dann geht das Wissen mit ihnen. Wenn eine gute Ausgangslage für die nächsten Aufträge geschaffen werden soll, sei Dokumentation unumgänglich.

5.6. Fazit

Crystal Clear enthält diejenigen Elemente, die bei Interviews mit Cockburn immer wieder als Erfolgsfaktoren hervorgehoben wurden:

- Nahe beieinander arbeiten und enge Kommunikation steht im Mittelpunkt
- Häufige Releases
- Informationen von den tatsächlichen Anwendern
- Werkzeuge zur Versionen-Kontrolle

Crystal Clear ist auf kleine Teams beschränkt. Es können zwar auch grosse Probleme angegangen werden, doch die Anzahl der Leute ist nicht beliebig erweiterbar. Dass in kleinen Teams besser kommuniziert werden kann, ist wohl keine neue Erkenntnis von Cockburn, oder dass mit einer guten Kommunikation leichtere Methoden möglich sind. Seine Kernaussage ist aber auch, dass es möglich ist, mit wenig Leuten ein Projekt durchzuführen, für das sonst mehr Leute vorgesehen wären.

Je kleiner das Team, desto leichter die Methode. Ich glaube, da streitet auch niemand darüber. Cockburn hat aber mehrere Male festgestellt, dass, obwohl man sich dieser Tatsache bewusst ist, trotzdem immer zu übergewichtige Methoden verwendet wurden. Insofern dienen die Crystal Methoden, sich ab und zu wieder mal daran zu erinnern, überflüssige Last über Bord zu werfen.

Cockburn bleibt in seinen Ausführungen bewusst unpräzise. Es ist ja auch nicht möglich, einen expliziten Prozess zu befolgen und gleichzeitig agil zu bleiben. Die Entscheidungen im Laufe eines Projekts müssen nun mal individuell gefällt werden, und es wäre wohl unrealistisch, auf jede mögliche Situation schon eine fixfertige Anleitung zu geben. Wenn aber ein Team wirklich etwas Neues ausprobieren möchte, stellt Crystal Clear eine gute Hilfestellung für den Anfang zur Verfügung. Was dann daraus gemacht wird, ist jedem selber überlassen.

6. Crystal Orange

6.1. Anwendungsbereich [2]

Während Crystal Clear Methoden für kleinere Projekte mit maximal 6 Arbeitskräften beschreibt, ist die Projektgrösse von Crystal Orange Projekten deutlich höher mit typischerweise 40 am Projekt beschäftigten Leuten. Um auch hier die Kommunikationswege möglichst kurz zu halten, sollen alle Leute im gleichen Gebäude beschäftigt werden. Durch den höheren Projektumfang bedingt, sind vor allem stärkere Koordinations- und Kontrollmechanismen als bei einem Crystal Clear Projekt nötig. Die typischen Charakteristiken eines solchen Projekts lassen sich kurz wie folgt zusammenfassen:

- total 10-40 Leute beschäftigt
- Projektdauer 1-2 Jahre
- Lieferfristen sind wichtig
- Kommunikation mit derzeitigen und zukünftigen Mitarbeitern ist wichtig, Zeit und Kosten sollen tief gehalten werden
- Das System ist nicht als “life critical“ klassifiziert

6.2. Strukturierung des Projektes [2]

A. Cockburn schlägt folgende Rollen von Personen vor, die am Projekt beteiligt sein sollen:

Geldgeber, Business Expert, Usage Expert, Technologievermittler, Business Analyst/Designer, Projektmanager, Systemarchitekt, Design Mentor, Leitender Designer-Programmierer, Designer-Programmierer, UI Designer, Tester

Die Angestellten werden in folgende Teams unterteilt:

- Systemplanung:
  Leute aus diesem Team nehmen einen Auftrag entgegen, halten die Anforderungen fest und erarbeiten einen Projektplan. Auf veränderte Anforderungen muss reagiert werden können.
- Projektüberwachung:
  Das Projekt muss laufend überwacht werden: Sind die Organisationsformen passend gewählt? Sind wir im Zeitplan? Wird das Budget eingehalten?
- Architektur:
  Das Architektur-Team ist für das Design des Systems verantwortlich. Auch hier kann es vorkommen, dass eine Architektur im Laufe des Projektes abgeändert werden muss, z.B. wegen veränderten Anforderungen oder Nichtrealisierbarkeit.
- Technologie:
  Technologien, die für das Projekt in Frage kommen, werden evaluiert und

ausgewählt. Schnittstellenkompatibilität soll von diesem Team auch abgeklärt werden.

- Funktionalität:
  Dieses Team ist für die Implementierung der Software zuständig. Interne Tests und Fehlerbehebung gehören auch in den Aufgabenbereich dieses Teams
- Infrastruktur:
  Welche Infrastruktur wird für das Projekt benötigt? Was muss neu beschafft werden? Wie werden die vorhandenen Ressourcen am besten aufgeteilt?
- Externe Tests:

  In dieses Team gehören auch Personen vom Auftraggeber. Nicht nur die Final-Release, sondern auch Prototypen und User Interfaces sollen von externen eingesehen werden, so dass Schwächen frühzeitig zum Vorschein kommen.

Hierbei muss allerdings noch festgehalten werden, dass eine am Projekt arbeitende Person in mehreren Teams gleichzeitig beschäftigt sein kann, zumal sich die Auslastung der einzelnen Teams während der Gesamtlaufzeit des Projektes stark verändern wird.

Ferner sollen während dem Entwicklungsprozess folgende Dokumente bzw. Arbeitserzeugnisse erstellt werden:

Anforderungsanalysen, Release Folgen, Projektplan, Statusberichte, UI Design Dokumente, Gemeinsames Objekt Modell, Team-interne Spezifikationen, Benutzerhandbuch, Source Code, Testfälle, Lauffähiges Programm

Für diese Dokumente existieren keine innerbetrieblichen Standards, stattdessen sollen sie soweit ausgearbeitet werden, bis sie von den Arbeitskollegen oder dem Auftraggeber als akzeptabel befunden werden.

Für alle Dokumente, die nur innerhalb eines Teams verwendet werden, legt das Team selbst die Standards fest, die Projektleitung erlässt hier keine Vorschriften.

6.3. Holistic Diversity – eine Gruppenorganisationsform [3]

6.3.1. Kurzübersicht

Im funktionalen Team, welches für die Implementierung der Software zuständig ist, fällt zweifellos der grösste Arbeitsaufwand am Projekt an. Hier ist eine weitere Aufteilung der Beschäftigten in Gruppen notwendig, um das Team in kleinere überschaubare Einheiten zu zerlegen.

6.3.2. Negativ-Szenario

Diese Strategie soll laut Cockburn angewendet werden, um eine “Throw it over the wall“-Entwicklung zu verhindern. In diesem Negativ-Szenario verrichtet jeder Beteiligte nur die ihm explizit zugewiesenen Arbeiten. Ein Systemarchitekt würde zum Beispiel ein UML-Diagramm erstellen. Sobald er die Arbeit erledigt hat – oder

das zumindest glaubt, wirft er das Resultat “über die Wand“ und hat danach nichts mehr damit zu tun. Der Systemarchitekt würde sein Diagramm also den Programmierern übergeben. Ist die Arbeit aber von mangelnder Qualität oder gar mit Fehlern behaftet, so muss die nächste Gruppe – im Beispiel also die Programmierer – dafür einstehen und die Fehler wieder korrigieren.

Es ist geradezu nahe liegend, dass bei einer solchen Organisation die Stimmung zwischen den Gruppen ziemlich schnell gegen den Nullpunkt steuern wird. Träten im genannten Beispiel Schwierigkeiten in der Entwicklung auf, so würden sich die beiden Gruppen gegenseitig der Inkompetenz beschuldigen.

6.3.3. Holistic Diversity im Detail

Jeweils 3-6 Leute sollen eine Gruppe bilden. Die Gruppen sollen aber nicht nach den Phasen der Entwicklung oder den Spezialgebieten der beteiligten Entwickler strukturiert werden. Der Grundgedanke der “Holistic Diversity“ (auf Deutsch in etwa: Ganzheitliche Vielfalt) besteht darin, möglichst viele verschiedene Spezialisten (System-Designer, UI-Designer, Business-Analysten, Datenbankspezialisten, Programmierer, ...) in einer Gruppe zusammenzuschliessen. Jede dieser Gruppen übernimmt dann die Verantwortung für ein Modul oder eine Teilfunktion des Projektes.

Vom Projektmanagement sollen nicht die Leistungen einzelner Personen bewertet werden, sondern nur die der Gruppen. Dadurch wird die Verantwortung der einzelnen Personen gefördert. Die Gruppen werden angespornt, selbst eine möglichst effiziente Nutzung ihrer Ressourcen (=Arbeitskräfte) anzustreben. Die Angestellten können sich nicht hinter ihren Spezialgebieten verstecken, sondern müssen, wo nötig, ihre Kollegen unterstützen, da sie beim Scheitern ihres Teilprojektes mit verantwortlich wären.

Das Projektmanagement setzt Richtlinien, wie die Gruppen ihre Endprodukte zu dokumentieren haben. Auf welche Art innerhalb einer Gruppe dokumentiert wird, bleibt ihr selbst überlassen.

Teillieferungen erfolgen nicht schriftlich, sondern mündlich. Somit hat der Empfänger eines Auftrags unmittelbar beim Erhalt die Möglichkeit, nachzufragen oder eventuell seine Bedenken zur vorhergehenden Arbeit zu äussern. Da eine Gruppe sehr nahe beieinander arbeitet, wenn möglich im selben Raum, wird die Kommunikation allgemein gefördert.

Die Grundgedanken der Holistic Diversity Strategie lassen sich durch die folgenden vier Leitsätze zusammenfassen:

- Kleine Gruppen fördern die Eigenverantwortung.
- Face-to-Face Kommunikation fördert schnelles Feedback.
- Einblick in die geleistete Arbeit fördert gegenseitigen Respekt.
- Multifunktionale Gruppen fördern individuelle Stärken.

6.4. Fazit

Ganz offensichtlich sind für ein Crystal Orange Projekt mit ca. 40 Beschäftigten im Gegensatz zu Crystal Clear mit 3-6 Leuten schwergewichtigere Methoden notwendig. Deshalb beschreibt A. Cockburn auch ziemlich genau, welche Rollen in einem solchen Projekt zu besetzen sind, in welchen Teams die Leute untergebracht werden sollen und welche Arbeitserzeugnisse quasi als Meilensteine zu erzeugen sind. Diese Ausführungen betreffen aber nur das Projekt im Grossen, man könnte hier von einer Makroorganisation sprechen.

Bei der “Holistic Diversity“ Strategie, die man als Mikroorganisationsform bezeichnen könnte, sind Parallelen zu Crystal Clear offensichtlich. Crystal Orange könnte also auch wie folgt umschrieben werden: Das Team wird in kleine Gruppen zerlegt, die nach Prinzipien von Crystal Clear organisiert sind; die Organisationsstrukturen, welche diese Gruppen zusammenhalten, machen dann Crystal Orange aus.

Leider war es nicht möglich einen Bericht über den Einsatz von Crystal Orange in der Praxis zu finden, der nicht von A. Cockburn selbst stammt. Dies führt bald zu Zweifeln, ob denn diese Methoden überhaupt irgendwo zur Anwendung kommen oder ob es sich um ein rein theoretisches Konzept handelt, das vorwiegend in Cockburns Kopf existiert. Die Antwort auf diese Frage könnte damit begründet werden, dass Crystal Orange wohl kaum 1:1 in die Praxis übernommen wird. Vielmehr handelt es sich um eine Art Leitfaden, wie ein Projekteam mit ca. 40 Mitarbeitern agile Softwareentwicklung betreiben kann. Die Methoden erfordern ein gewisses Tuning, um sie auf die individuelle Situation abzubilden. Auch wäre denkbar, dass eine Firma nur gewisse Teilaspekte von Crystal Orange übernimmt. In einem solchen Fall könnte wohl nicht eindeutig festgehalten werden, ob nun “Crystal Orange“ angewendet wird oder nicht. Die Grenzen sind nicht so strikt, wie dies bei gewissen ISO-Zertifizierungen der Fall ist.

Die folgende Fallstudie “Crystal Orange Web“ soll nun einen Aufschluss darüber geben, wie Crystal Orange in der Praxis modifiziert zum Einsatz kommen kann. Teilweise kann es ziemlich schwierig sein von der Fallstudie noch konkrete Parallelen zum theoretischen Ansatz von Crystal Orange festzustellen.

7. Crystal Orange Web

7.1. Situationsanalyse [2]

Crystal Orange Web umfasst Methoden, die von A. Cockburn „massgeschneidert“ für die Firma eBucks.com entwickelt wurde.

eBucks.com entwickelt und unterhält im Auftrag von Banken Finanz- und Onlineshoppingportale. Die eigens entwickelte Währung, den eBuck, erhalten die Endkunden als Bonuspunkte für getätigte Geschäfte oder zu einem bestimmten Kurs gegen Bargeld.

Der Hauptunterschied zu Crystal Orange besteht darin, dass eBucks.com nicht einzelne von einander abgrenzbare Projekte realisiert, sondern seinen Kunden in einem kontinuierlichen Strom von Kleinaufträgen Code abliefert. Dieser Code wird wiederum in bereits bestehende Plattformen eingebunden und so den Benutzern zugänglich gemacht.

Auch aus dem folgenden Grund war eBucks.com in einer für A. Cockburn interessanten Situation: Die Firma hatte sich bereits im Internet verfestigt. Das Vordringen in einen neuen Markt war also nicht mehr das primäre Ziel. Das grösste Problem waren vielmehr die vielen Reklamationen, die in stark steigender Tendenz eintrafen. Die Firma verlagerte also ihre grösste Aufmerksamkeit von der Produktivität zur Behebung von Störfällen.

Bei eBucks.com arbeiteten ungefähr 50 Leute aus verschiedensten Berufsgruppen. Cockburn klassifizierte die Firma als eine E50 Situation: Beim Scheitern eines Projektes kann eine beträchtliche (essential) Menge Geld verloren gehen und insgesamt sollen etwa 50 Leute miteinander kooperieren können.

Nachdem Cockburn in allen Geschäftsbereichen Angestellte interviewet hatte, kam er zu den folgenden Tatsachen:

- Die Informationsströme waren ziemlich gut. Alle arbeiteten auf dem selben Stockwerk. Es waren verschiebbare Glas und Whiteboard Wände vorhanden, so dass die Angestellten nahe beieinander waren aber trotzdem eine gewisse Privatsphäre hatten.
- Jede Person arbeitete an mehreren Projekten und konnte sich deshalb nie auf eine bestimmte Arbeit konzentrieren. Die Angestellten wurden ständig durch irgendwelche Anfragen von ihrer eigentlichen Arbeit abgelenkt.
- Einstellung, Freundschaftlichkeit und Arbeitsmoral waren ziemlich hoch, wegen den vielen Unterbrüchen und dem stagnierenden Fortschritt der Firma aber im Sinken begriffen. Zudem hatten die Programmierer ihre Arbeitsplätze auf der einen Seite des Gebäudes und die Angestellten der Administration auf der anderen. Dies führte immer mehr zu abschätzigem Gerede der einen Gruppe über die andere.
- Den Flaschenhals ortete Cockburn bei den Programmierern: Diese waren mit ihrer Arbeit dauernd im Rückstand, da sie sich kaum auf ihre Kernaufgaben

konzentrieren konnten. Ständig mussten sie Auskunft über den Fortschritt ihrer Arbeit geben oder wurden mit sonstigen Anfragen überhäuft.

- eBucks.com war noch nicht ein Jahr alt. Deshalb konnte sich noch keine grosse Firmentradition entwickeln; die Angestellten waren also immer noch sehr offen gegenüber Neuerungen.

7.2. Kurze Beschreibung von Crystal Orange Web [2]

Die Methoden wurden als Menge von Übereinkünften und in fünf Kategorien gegliedert festgehalten:

7.2.1. Regular Heartbeat, with Learning

Das Ziel dieser Kategorie ist die Einführung eines Haupt Geschäftsablaufs, der es ermöglichen soll, in regelmässigen Abständen ein Feedback über die geleistete Arbeit zu erhalten. Die Angestellten sollen über die vergangene Arbeit und wo Verbesserungen möglich sind nachdenken können.

Zweiwöchige Entwicklungszyklen: Im Gesamten läuft die Produktion in fixen Entwicklungszyklen von zweiwöchiger Dauer ab. Am Ende der Zyklen kann jedes Team für sich auswählen, ob der nächste Zyklus zwei oder vier Wochen dauern soll. Jedes Team muss mindestens alle vier Wochen ein brauchbares (Zwischen-) Produkt abliefern.

"Reflection Workshops" am Ende der Zyklen: Am Ende jedes Zyklus kommen die Angestellten zusammen, um zu besprechen, was gut und was schlecht funktioniert hat. Auch werden neue Ideen, die im kommenden Zyklus umgesetzt werden sollen, vorgebracht. Das Ergebnis des Workshops wird als Leitfaden für den nächsten Zyklus schriftlich festgehalten.

7.2.2. Basic Process

Diese Kategorie beinhaltet Punkte über innerbetriebliche Abläufe bei eBucks.com. Wer erledigt welche Arbeiten? Wer fällt welche Entscheidungen? Diese Fragen sollen beantwortet werden, mit dem Ziel, eine weitsichtige langfristige Planung zu ermöglichen. Da die Punkte sehr stark auf die Auftragsabwicklung bei eBucks.com massgeschneidert sind, will ich nicht im Detail auf alle eingehen.

Durch einen kurzen Anwendungsfall werden neue gewünschte Systemeigenschaften im Einsatz beschrieben.

Techniker schätzen den Implementierungsaufwand anhand des Anwendungsfalls. Darauf muss das Projekt vom Management bewilligt werden.

Detaillierte Anwendungsfälle werden erstellt. Die Programmierer implementieren die neuen Module. Durch Reviews und Regressionstests sollen Fehler eruiert werden. Nach der Integration der neuen Module werden die ganzen Systeme nochmals durch Integrationstests getestet.

Die Veränderungen am System werden bekannt gegeben und dem Call Center detailliert mitgeteilt.

Das Call Center meldet entdeckte Fehler einem SWAT Team, dessen einzige Aufgabe die Fehlerbehebung ist. Dieses SWAT Team wird durch einen Rotationsmodus von den Entwicklern gestellt.

7.2.3. Maximum Progress, Minimum Distraction

Der Zweck dieser Kategorie ist sicherzustellen, dass die Angestellten an den für die Firma wichtigsten Projekten arbeiten können. Auch soll ihnen ermöglicht werden, sich ohne ständige Ablenkung auf diese Arbeiten konzentrieren zu können.

Die wichtigsten Aufgaben werden für jeden zweiwöchigen Zyklus schriftlich festgehalten. Zudem werden diese Arbeiten den Angestellten zugeordnet, so dass jeder seine wichtigsten Bereiche für einen Zyklus kennt.

Jede Person, die an mehreren Projekten arbeitet, erhält während jedem Zyklus mindestens zwei Tage pro Projekt, um ungestört daran arbeiten zu können.

Die Entwickler haben auf der Aussenseite ihrer Büros Whiteboards aufgestellt. Darauf halten sie den Status ihrer Arbeit sowie den Wochenplan fest. Jeden Morgen findet ein kurzes Meeting zwischen den Entwicklern und Projektleitern statt. Der Projektleiter darf während dem Rest des Tages keine Anfragen mehr über den Fortschritt der Arbeit stellen. Die Zeit zwischen 10 und 12 Uhr wird zur “Focus Time“ erklärt. Während dieser Zeitspanne finden keine Meetings statt und üblicherweise werden die Telefone ausgeschaltet.

7.2.4. Maximally Defect Free

Das Ziel dieser Kategorie ist die Fehlerbehebung.

Jeder Programmierer schreibt für seinen eigenen Code vor dem Entwickeln eine Reihe von Testfällen, auf die der Code ständig geprüft wird. Der Code wird erst ins System integriert, wenn ein zweiter Programmierer den Code eingesehen und für dessen Qualität gebürgt hat.

Der Server stellt eine spezielle Funktion zur Verfügung, so dass die Integrationstester ihre eigene Testdatenbank unterhalten können. Somit wird die Arbeit letzterer erleichtert und die Effizienz gesteigert. Mit Hilfe einer eigenen rudimentären Programmiersprache werden Beispieltransaktionen und erwartete Antworten konstruiert. Somit können auf einfache Weise Testszenarien erstellt und in die Datenbank aufgenommen werden. Die Arbeit der Tester wird erleichtert und die Effizienz gesteigert.

Im Call Center werden “Screen-Click Statistiken“ erstellt und an die Entwickler weitergegeben. So kann rasch erkannt werden, wo für die Benutzer Probleme bestehen (entweder durch unklare Navigation oder Fehler im System).

7.2.5. A Community, Aligned in Conversation

In dieser Kategorie soll das langfristige Ziel, auf das die Firma zusteuert, festgehalten werden.

Cockburn schlägt vor, dass themenübergreifende Teams aus den verschiedenen Berufsgruppen gebildet werden sollen (Programmierer, UI Designer, Tester,

Projektleiter, Marketing Fachleute, usw.). Ziel ist es, die Kommunikation zwischen den einzelnen Teilbereichen noch stärker zu fördern und abschätzige Ansichten der einen Berufsgruppe gegenüber einer anderen zu vermeiden. Cockburn hätte also gerne seine “Holistic Diversity Strategy“ umgesetzt.

7.3. Sechs Monate später [2]

Die Methoden wurden so vorgestellt, wie sie A. Cockburn unmittelbar nach der Situationsanalyse konstruiert hatte. Selbstverständlich kann nicht erwartet werden, dass alle Punkte in der Firma genau wie von Cockburn vorgeschlagen umgesetzt wurden. Eine Betrachtung nach sechs Monaten soll darüber Aufschluss geben, wie erfolgreich Cockburns Praktiken angewandt wurden.

Folgende Punkte wurden bei eBucks.com immer noch erfolgreich angewendet:

- **Der vierzehntägige Herzschlag** erwies sich als besonders hilfreich. Entwickler und Projektleiter wurden dazu gedrängt, ihre Arbeit zu planen, Tester erhielten die Möglichkeit strukturiert zu testen und Kunden konnten regelmässig über geplante Upgrades informiert werden. Ein dreiwöchiger Herzschlag wurde ebenfalls in Erwägung gezogen, dann aber als zu lang empfunden. Komplexe Projekte wurden teilweise auch über eine vierwöchige Periode geplant, zwei Wochen aber stark bevorzugt. Die Meetings am Ende des Heartbeats wurden ebenfalls strikt eingehalten. Der CEO nutzte diese Gelegenheit um seine Anliegen vor der ganzen Firma vorbringen zu können. Gemachte Fehler wurden besprochen und Vorschläge eingebracht, was man in der Zukunft verbessern kann.
- Durch das **Vetorecht der Tester** hat sich die Qualität des freigegebenen Codes deutlich verbessert. Obwohl dies für die Programmierer nicht immer angenehm war...
- Die **Focus Time** konnte sich auch durchsetzen und hat sich als sehr wichtig erwiesen. Jeden Morgen wurde um 10 Uhr eine Glocke geläutet und sogar der CEO selbst meinte, er kriege eine Panik-Attacke, wenn er diese zwei Stunden nicht für sich habe.
- Das **SWAT Team** kam immer wieder zum Einsatz und konnte nach Anfragen aus dem Call Center wiederholt Fehler in veröffentlichtem Code ausfindig machen.

Folgende Vorschläge konnten sich leider nicht durchsetzen und wurden nach sechs Monaten kaum mehr praktiziert:

- Die Entwickler benutzten die Whiteboards vor ihren Büros kaum, d.h. sie hielten ihren Fortschritt und ihre Prioritäten nicht darauf fest. Vielleicht werden sie nicht mehr von ständigen Anfragen über ihre Arbeit überhäuft oder vielleicht sind sie auch einfach nur zu faul.
- Die meisten Entwickler arbeiten an maximal drei Projekten zur selben Zeit. Ausser zwei Leuten an Schlüsselpositionen, die bis zu 15 Aufgaben auf ihrer Liste haben. Der Rückstand dieser Leute stiess bei den anderen Entwicklern immer wieder auf Ärger, da sie auf deren Arbeit angewiesen waren. Die

> Ursache dieses Problems versuchte der CEO auf ein altbekanntes Phänomen zurückzuführen: Die richtige langfristige Lösung, andere Leute mit Teilen der Arbeit zu beauftragen und auszubilden, würde kurzfristig länger dauern als diese Arbeiten rasch selbst zu erledigen.

7.4. Fazit

eBucks.com war eine innovative Firma, die nach einem beachtlichen Anfangserfolg ins organisatorische Chaos zu stürzen drohte: Zuständigkeiten waren kaum klar geregelt, die Arbeit der verschiedenen Teams war schlecht aufeinander abgestimmt. Cockburns Auftrag war es, diese Missstände zu beheben und die Firma durch ein geeignetes Reorganisationskonzept langfristig zu unterstützen. Glaubt man seinen Ausführungen, insbesondere der Analyse nach 6 Monaten, so scheint seine Arbeit ein Erfolg zu sein. Dies soll aber hier nicht weiter in Frage gestellt werden.

Das Projekt kann aber von einer anderen Seite durchleuchtet werden: Cockburn beschreibt Crystal Orange Web als eine Mutation von Crystal Orange massgeschneidert für kurze Entwicklungszyklen. Dabei bleibt die Frage offen, inwiefern Zusammenhänge zwischen Crystal Orange und Crystal Orange Web bestehen. Die Organisation der Mitarbeiter in Teams war bei eBucks.com schon vorgegeben und Cockburn liess es dabei bestehen. Die Holistic Diversity Strategie kam nach seinen eigenen Angaben (leider) nicht zur Anwendung und die Prozessabläufe werden in Crystal Orange nicht so detailliert wie in Crystal Orange Web beschrieben. Parallelen zu Crystal Orange sind also schwer nachvollziehbar.

Man kann sogar noch weiter gehen und nach Übereinstimmungen zwischen Crystal Orange Web und Agilen Methoden allgemein suchen. Diese finden sich vor allem in den Punkten “Regular Heartbeat, with Learning“ und “A Community, Aligned in Conversation“, wobei aber letzterer gar nicht wirklich umgesetzt wurde. Die anderen Punkte beinhalten eher Aspekte der innerbetrieblichen Organisation, die so sehr auf die hier vorliegende Firma abgestimmt sind, dass ihre Übereinstimmung zu Agilen Methoden kaum mehr untersucht werden kann.

Gewisse Punkte von Cockburns Organisationsformen finde ich persönlich genial. Ich möchte aber doch noch die Frage aufwerfen, wie die Firma wohl organisiert wäre, hätte man nicht bewusst auf agile, sondern auf klassische Methoden der Softwareentwicklung gesetzt? Wäre da vieles anders?

8. Literatur- und Quellenverzeichnis

[1] http://alistair.cockburn.us/me/alistaircv.html

[2] Cockburn, A.: *Agile Software Development*, 2002, Addison Wesley

[3] http://members.aol.com/acockburn/riskcata/riskbook.htm:

[4] http://members.aol.com/humansandt/crystal/clear/ (Entwurf des Buches „Crystal Clear“ von Alistair Cockburn, noch nicht erschienen)

[5] Manuel Meyer: Extreme Programming

[6] Highsmith, J.: Adaptive Software Development, 2000, Dorset House, New York