Hochschule für Technik und Wirtschaft Dresden (FH)
Fachbereich Informatik/Mathematik

# Diplomarbeit

**im Studiengang Wirtschaftsinformatik**

| | |
|---|---|
| **Thema:** | Emergenz in der Softwareentwicklung – bereits verwirklicht oder Chance? |
| **eingereicht von:** | Sebastian Stein <emergenz@hpfsc.de> |
| **eingereicht am:** | 26. Mai 2004 |
| **Betreuer:** | Frau Prof. Dr.-Ing. S. Hauptmann |

# Inhaltsverzeichnis

## Abbildungsverzeichnis

# 1 Einleitung

## 1.1 Motivation

Seit über 35 Jahren[1] findet eine wissenschaftliche Auseinandersetzung mit dem Gebiet der Softwareentwicklung statt. Seit den Anfängen konnten bereits bemerkenswerte Fortschritte in den drei Kernbereichen

- Qualität
- Produktivität
- Kosten

erzielt werden. Gleichzeitig stiegen die Anforderungen an die Softwareentwicklung – Softwaresysteme werden immer komplexer. Diese Entwicklung geht zumindest genauso schnell vonstatten, wie die Verbesserungen in der Softwareentwicklung.

Heutzutage wird Softwareentwicklung als ein ingenieurtechnischer Prozess verstanden und teilweise als solcher umgesetzt. Trotz einer Erhöhung der Komplexität und des Umfangs der Planungsmethoden und Vorgehensmodelle ist die resultierende Software in Hinblick auf Qualität, Einhaltung des Entwicklungsbudgets und termingerechter Auslieferung oft mangelhaft. Der durch immer komplexere Vorgehensmodelle benötigte Aufwand steht in keinem Verhältnis zu den Verbesserungen durch diese Vorgehensmodelle.

## 1.2 Ansatz

Die vorliegende Arbeit unternimmt deshalb den Versuch, die Problematik der Softwareentwicklung aus einer anderen Perspektive darzustellen. Damit verbindet sich die Hoffnung, die grundlegenden Probleme der Softwareentwicklung identifizieren zu können. Durch die Betrachtung der Problematik aus dem Blickwinkel der Emergenz wird versucht, neue Ansätze für die Softwareentwicklung zu finden bzw. vorhandene Ansätze zu vereinheitlichen. Einschränkend muss erwähnt werden, dass diese Arbeit nicht den Versuch unternimmt einen „Silver-Bullet“ (vgl. Bro01a) zu formulieren. Mit der neuen Sichtweise Emergenz in der Softwareentwicklung wird auch weiterhin lediglich eine moderate Produktivitäts- und Qualitätssteigerung sowie Kostenreduzierung erreicht werden können.

Als Ausgangspunkt für diese Arbeit bietet sich die Untersuchung des Begriffs Emergenz an. Anhand der Definition wird untersucht, welche Modelle zur Erklärung von Emergenz existieren. Dies wird in der Endkonsequenz bis zur Frage der philosophischen Grundauffassung – dem Weltbild – führen.

[1] Als Ausgangspunkt wird die NATO Konferenz zum Thema Software Engineering im Jahr 1968 in Garmisch angesehen.

Weiterhin ist von Interesse, ob und wie bereits Emergenz in anderen Bereichen der Wirtschaftsinformatik eingesetzt wird. Bevor diese Erkenntnisse auf die Softwareentwicklung übertragen werden können, muss zunächst die aktuell praktizierte Softwareentwicklung nachgezeichnet werden, damit ein Vergleich möglich wird. Anschließend kann untersucht werden, ob Emergenz in der Softwareentwicklung anwendbar ist, vielleicht schon angewendet wird und ob eine Anwendung eine Chance bietet.

## 1.3 Abgrenzung

Diese Arbeit wählt bewusst einen interdisziplinären Ansatz, ganz im Sinne der Wirtschaftsinformatik. Es wird versucht eine Brücke zwischen Naturwissenschaft und Gesellschaftswissenschaft zu schlagen. Die Softwareentwicklung konnte schon mehrfach von der Adaption von Konzepten aus anderen Wissenschaften profitieren. Erwähnt sei an dieser Stelle z. B. die ursprünglich aus der Architektur stammenden Entwurfsmuster nach Gamma. (Gam96)

Da im Rahmen dieser Arbeit teils sehr unterschiedliche Theorien vorgestellt werden, müssen entsprechende Vereinfachungen vorgenommen werden und die meisten erwähnten Anwendungen können nur relativ kurz diskutiert werden. Dies wird nicht als Nachteil angesehen, da oftmals in vielen Bereichen ein hohes Detailwissen existiert, eine bereichsübergreifende Kombination findet allerdings zu selten statt. Der Bezug zum Detailwissen wird mittels eines umfangreichen Literaturverzeichnis sowie Fußnoten zur Verfügung gestellt.

Weiterhin findet keine kritische Bewertung einer möglichen Anwendung von Emergenz in der Softwareentwicklung statt. Dies würde den Umfang dieser Arbeit sprengen und sollte in einer eigenen Arbeit näher untersucht werden.

# 2 Emergenz

## 2.1 Begriff und Definition Emergenz

Bevor auf den Begriff Emergenz näher eingegangen werden kann, muss der Systembegriff kurz charakterisiert werden, da Emergenz in Systemen auftritt. Ein System ist definiert als „eine Menge von Elementen, die miteinander durch Beziehungen verbunden sind und gemeinsam einen bestimmten Zweck zu erfüllen haben.“ (Mül00, S. 48) Somit besteht ein System aus *Systemelementen*, Beziehungen zwischen den Elementen, den so genannten *Relationen*, und einer *Systemgrenze*. Mit der Systemgrenze kann klar zwischen System und seiner *Umwelt* unterschieden werden, die Systemgrenze grenzt das System von der Umwelt ab.

In dieser Arbeit steht der Begriff *aktueller Systemzustand* für die Zusammenfassung von allen Systembestandteilen sowie die sich daraus ergebende *Systemstruktur* und *Systemordnung*. Unter einer Änderung des Systemzustandes wird somit die Änderung von Systemelementen, Relationen, Systemgrenze, Systemstruktur oder Systemordnung verstanden.

In der Physik bezeichnet man ein System als *offen*, wenn ein Energieaustausch zwischen Umwelt und System stattfindet. In einem *geschlossenen* System hingegen findet kein Energieaustausch mit der Umwelt statt und deshalb gilt der zweite Hauptsatz der Thermodynamik.[2] Die in dieser Arbeit betrachteten Systeme können alle als offen bezeichnet werden. In sozialen Systemen entspricht der Energieaustausch dem Informationsaustausch mit der Umwelt.

Das Substantiv Emergenz kann auf das lateinische Verb emergo zurückgeführt werden. Das Verb emergo hat im Deutschen die Bedeutung „auftauchen“ und „sich herausarbeiten“. (vgl. Men98) Die Verwendung des Wortes kann bis zum Anfang unserer Zeitrechnung zurückverfolgt werden (vgl. Geo13)[3].

Aus der Bedeutung des lateinischen Verbes emergo lässt sich die Bedeutung des Wortes Emergenz ableiten. Man versteht demzufolge unter Emergenz das Auftauchen von Systemzuständen, die nicht durch die Eigenschaften der beteiligten Systemelemente erklärt werden können. Im Duden findet man die Definition von Emergenz als das Phänomen „wonach höhere Seinsstufen durch neu auftauchende Qualitäten aus niederen entstehen“. (Dud00) Bei dieser Definition ist zu beachten, dass die „neu auftauchenden Qualitäten“ erst „entstehen“ und nicht bereits vorhanden sind. Im Volksmund wird diese Idee formuliert als: „Das Ganze ist mehr als die Summe seiner Teile.“ Für dieses „Mehr“ bzw. dessen Entstehung steht der Begriff Emergenz.

Das Phänomen der Emergenz lässt sich am Beispiel Temperatur ver-

[2]Erklärung zum zweiten Hauptsatz der Thermodynamik siehe Kapitel 3.2 auf Seite 11
[3]z. B. bei Velleius Paterculus in „Historia Romana“ ca. 29 n. Chr.

deutlichen. Betrachtet man ein einzelnes chemisches Molekül, wie z. B. das Wassermolekül, dann kann man für dieses Molekül keine Temperatur bestimmen. Hat man allerdings eine große Menge des einzelnen Moleküls, dann ist es möglich eine Temperatur zu ermitteln. Die Temperatur entsteht erst, wenn viele Moleküle aufeinander treffen. Somit kann die Temperatur als eine emergente Eigenschaft vieler Moleküle angesehen werden. Bei Wasser ist die Temperatur eine emergente Eigenschaft der Wassermoleküle, aber es ist keine emergente Eigenschaft des Wassers.

## 2.2 Emergenz als Prozess

Ausgehend vom Systembegriff kann gesagt werden, dass ein System den aktuellen Systemzustand A besitzt. Wird dieses System in einen neuen Systemzustand B überführt, findet eine *Transformation* (vgl. Ess00) von Systemzustand A nach Systemzustand B statt. Diese Transformation ist das Ergebnis eines *Transformationsprozesses*. Man bezeichnet den Transformationsprozess als Emergenz, wenn der Systemzustand B nicht direkt aus Systemzustand A gefolgert[4] werden kann. Eine andere Bezeichnung für diesen Prozess liefert Fliedner (Fli99) mit dem Begriff *Komplexionsprozess*: „Da dieser Prozeß eine Komplexitätsebene verläßt und eine neue Komplexitätsebene anstrebt, nennen wir ihn *Komplexionsprozeß*.“ (Fli99, S. 12)[5] In dieser Arbeit wird der Begriff Transformationsprozess verwendet, da er im Rahmen der aus der Informatik bekannten Systemtheorie geläufiger ist.

Durch die Betrachtung von Emergenz als Transformationsprozess (nach Ess00) kann der Emergenzbegriff entmystifiziert werden. Dies eröffnet die Möglichkeit wissenschaftlicher Untersuchungen, welche „Transformationsregeln“ (Ess00, S. 13) konkret wirken. Der vollständige Zusammenhang lässt sich wie in Abbildung 1 auf Seite 9 veranschaulichen. Tatsächlich ist es so, dass bei Transformationen, deren Regeln vollständig bekannt sind, niemand von Emergenz sprechen würde. Somit ist Emergenz keine feste Größe, sondern immer vom aktuellen Kenntnisstand abhängig. (Ess00, S. 4) Bei erhöhtem Kenntnisstand spricht man dann unter Umständen nicht mehr von Emergenz.

## 2.3 Historizismus und Reduktionismus

Während des Transformationsprozesses treten neue Qualitäten auf, die nicht auf die Aufsummierung der Einzeleigenschaften zurückgeführt werden können. Aus diesem Ansatz ergibt sich die Frage, ob das Emergenzphänomen sich überhaupt auf einfache Transformationsregeln reduzieren lässt. Der Historizismus verneint diese These und behauptet im Gegenteil, dass es Gesetze auf höheren Ebenen (Makroebene) gibt, die sich nicht auf die Naturgesetze

[4]genaue Erläuterung dazu in Kapitel 3.5 auf Seite 19
[5]Hervorhebungen im Original

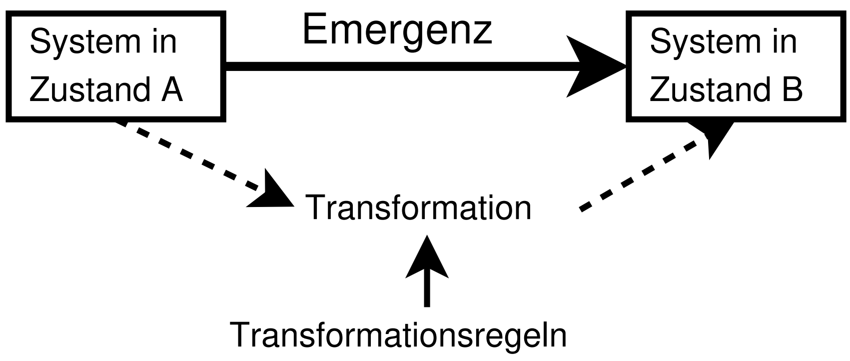


Abbildung 1: Emergenz als Transformationsprozess

reduzieren lassen. Zur Erklärung werden dann „Vitalkräfte und spirituelle Agentien“ (PJS94, S. 30) bemüht, die die physikalischen Naturgesetze außer Kraft setzen. Solch ein Makrogesetz müsste allerdings den Zustand aller Elemente genau beschreiben. Ein derart umfassendes Makrogesetz nimmt demnach keine Vereinfachung vor und ist somit selber die Realität. In diesem Zusammenhang von einem Gesetz oder einem Modell zu sprechen, erscheint sinnlos. Anhand dieser Argumentationslinie (nach Pop87)[6] wurde gezeigt, dass ein reiner Historizismus allgemein wenig sinnvoll ist.

Die Gegenströmung zum Historizismus ist der Reduktionismus. Reduktionisten stellen die These auf, dass jedes Phänomen auf wenige Naturgesetze reduziert werden kann. Im oben genannten Beispiel von Temperatur als emergenter Eigenschaft von Molekülen scheint dies anhand von Bewegungsgesetzen und dem ersten Hauptsatz der Thermodynamik durchaus möglich. Allerdings ist fraglich, ob jemals eine Reduzierung von menschlichem Gruppenverhalten auf die Bewegung von Atomen möglich ist. Eine Reduzierung aller Phänomene im Universum auf wenige Naturgesetze scheint sehr theoretisch. (vgl. Stö94) Der Ansatz ist in der Praxis wenig gangbar. Sinnvoller ist zu akzeptieren, dass eine Reduktion von Emergenz auf physikalische Naturgesetze (momentan) nicht möglich ist. Es müssen somit Erklärungsmodelle gefunden werden, die auf einer höheren Ebene ansetzen. Diese Erklärungsmodelle, wie im nächsten Kapitel vorgestellt, müssen unter Beachtung der bekannten Naturgesetze aufgestellt werden, um Historizismus zu vermeiden. (PJS94, S. 30ff) Aus dieser Forderung leitet sich das Anliegen dieser Arbeit ab. Es wird untersucht, wie die im nächsten Kapitel vorgestellten Erklärungsmodelle praktisch in der Software Entwicklung angewendet werden bzw. angewendet werden können. Es wird allerdings nicht untersucht, weshalb die Erklärungsmodelle wirken - eine Reduzierung, z. B. auf soziologische Theorien, findet nicht statt.

[6]speziell Kapitel 23: Kritik des Holismus

## 2.4 Zusammenfassung Begriff Emergenz

Die bisherigen Erläuterungen lassen sich folgendermaßen zusammenfassen:

1. Ein System kann seinen Zustand qualitativ ändern.
2. Die Änderung kann unter Umständen nicht auf die Eigenschaften der einzelnen Systemelemente und deren Relationen untereinander zurückgeführt werden.
3. Der neue Systemzustand ist somit mehr als eine reine Aufsummierung der Einzeleigenschaften der Systemelemente und deren Beziehungen.

Dieses Phänomen wird als Emergenz bezeichnet.

Das System, bestehend aus den Systemelementen, den Relationen und der Systemgrenze, wird in einem Transformationsprozess bzw. Komplexionsprozess verändert. Im nun folgenden Abschnitt wird untersucht, welche Erklärungsmodelle für diesen Transformationsprozess existieren. In diesem Zusammenhang wird der Begriff *Selbstorganisation* eingeführt, der von verschiedenen Autoren zur Erklärung des Emergenzphänomens verwendet wird.

# 3 Transformationsprozess

## 3.1 Überblick

Im nun folgenden Abschnitt werden drei Erklärungsmodelle für den Transformationsprozess vorgestellt:

- Synergetik nach Haken
- Chaostheorie
- Autopoiesis nach Maturana

Am Ende des Kapitels werden die drei Erklärungsmodelle auf die Begriffe *Nicht-Linearität*, *Nicht-Determinismus* und *Selbstorganisation* reduziert.

## 3.2 Erklärungsmodell Synergetik

Als erstes Erklärungsmodell wird die Wissenschaftsdisziplin Synergetik kurz umrissen. Synergetik ist die „Lehre vom Zusammenwirken“ (Hak91, S. 17), entwickelt in den 60er Jahren von Hermann Haken. Zu dieser Zeit entdeckte er die Lasertechnik. Dabei war von Interesse, warum sich die an einer diffusen Lichtquelle ausgestrahlten verschiedenen Lichtwellen zu einer Lichtwelle bündeln und sich dadurch der Laserstrahl (monochromatisches Licht) bildet. Anders formuliert lässt sich fragen, warum kommt es unter verschiedenen Lichtwellen zu einer *Selbstorganisation* mit dem Ergebnis einer einzigen Lichtwelle? Anhand dieser Frage ergibt sich die Definition des Begriffes Selbstorganisation. In einem System spricht man von *Selbstorganisation*, wenn ein Systemzustand einzig von den Systemelementen und den Relationen hervorgerufen wird, ohne Einfluss der Umwelt auf das System.

Die Synergetik versteht sich als eine fachübergreifende Wissenschaftsdisziplin (Hak91, S. 21) ähnlich der Mathematik und Statistik. Haken betont, dass die Synergetik nicht nur auf die Naturwissenschaft (z. B. Physik) angewendet werden kann, sondern eine Anwendung z. B. in Gesellschaftswissenschaften wie der Soziologie ebenfalls möglich ist. Für die Synergetik wurde von Haken ein umfangreiches mathematisches Formelwerk (vgl. Hak90) entwickelt. Danach kann die Synergetik verstanden werden als ein „Konzept zur Erklärung von Ordnungsbildung in stochastischen System mit vielen interagierenden Einheiten“. (Pas92, S. 3) An dieser Stelle wird auf eine mathematische Darstellung verzichtet und es folgt eine sprachliche Darstellung.

Ausgangspunkt der Betrachtungen zur Synergetik ist die Frage, warum sich im Universum komplexe Strukturen entwickeln konnten, wenn allein der zweite Hauptsatz der Thermodynamik (Hak91, S. 25ff) gilt? Der zweite Hauptsatz besagt, dass in einem geschlossenem System die Unordnung stets zunimmt. So findet z. B. in einem beheiztem Wasserkessel solange ein Wärmeaustausch statt, bis alle Wassermoleküle gleichmäßig verteilt sind und kein Temperaturunterschied innerhalb des Wasserkessels mehr besteht. Dieser Vorgang ist nicht umkehrbar, er ist *irreversibel*. Man wird nicht beobachten, dass sich plötzlich wieder Temperaturunterschiede in einem Wasserkessel ergeben.

Trotz des zweiten Hauptsatzes der Thermodynamik hat sich aber im Universum eine Vielzahl von bemerkenswerten Strukturen herausgebildet, wie z. B. das Leben auf unserer Erde. Es ist sehr unwahrscheinlich, wenn dies lediglich Zufall sein sollte. Haken untersucht deshalb mit der Synergetik, wie sich eine große Anzahl von Einzelelementen zu höheren Strukturen selbstorganisieren.

Da in der Synergetik eine Reihe von neuen Begriffen eingeführt werden, erfolgt die Darstellung an einem Beispiel (nach Pas92, S. 6). Dazu stelle man sich eine stark frequentierte Treppe in einem Gebäude vor. Die Treppe wird von vielen Menschen in beiden Richtungen genutzt. Am Anfang erwartet man, dass alle Menschen wild durcheinander gehen und sich dadurch gegenseitig behindern, wie in Abbildung 2 Teilbild a) auf Seite 13 gezeigt. Nun kann es passieren, dass einige Menschen der Spur ihres Vordermanns folgen. Aus diesem spontanen Verhalten kann eine Ordnung entstehen, wenn mehrere Menschen dem Beispiel folgen und ebenfalls die Spur für ihre Richtung wählen. Jetzt wird diese Spur für die Bewegung der Menschen zu einer Art Vorgabe (Abbildung 2 Teilbild b)). Haken bezeichnet dies als *Ordner*. Der Ordner zwingt die Individuen sich dieser Ordnung anzupassen. In der Sprache der Synergetik *versklavt* der Ordner die Individuen. Durch diesen Vorgang ist aus einer vorher völlig zufälligen Bewegung der Menschen auf der Treppe eine geordnete Bewegung entstanden (Abbildung 2 Teilbild c)). Der Ordner existierte nicht von Anfang an, sondern hat sich spontan aus dem Verhalten der Individuen ergeben. Haken spricht hier von einem *Phasenübergang*. In der Folge hat der Ordner einen starken Einfluss auf das Verhalten der Individuen. Dadurch ergibt sich ein geschlossener Regelkreis, da der Ordner erst aus dem Verhalten der Individuen hervorgegangen ist, nun aber das Verhalten der Individuen entscheidend beeinflusst. Dies bezeichnet man als *Zirkularität*.

Haken formuliert, dass sich durch die *Versklavung* der Individuen durch den Ordner ein Phasenübergang bildet. Während des Phasenübergangs zeigen sich bereits Eigenschaften von beiden Phasen. Allerdings besteht keine Kausalität zwischen den Phasen. Es kann nicht vorhergesagt werden, welcher neue Zustand durch den Ordner hervorgerufen wird. Am Beispiel der Treppe ist es in Deutschland sehr wahrscheinlich, dass sich „Rechtsverkehr“ ergibt. Allerdings ist das nicht zwangsläufig. Schon wenige englische Touristen auf der Treppe reichen aus, um vielleicht einen Ordner für „Linksverkehr“ zu bilden. Haken versteht unter *Nicht-Linearität*, dass kleinste Änderungen der Systemstruktur – eine *Fluktuation* – riesige Auswirkungen auf den Systemzustand haben können.

Durch die Ordner findet eine Komplexitätsreduzierung statt. Es ist nicht

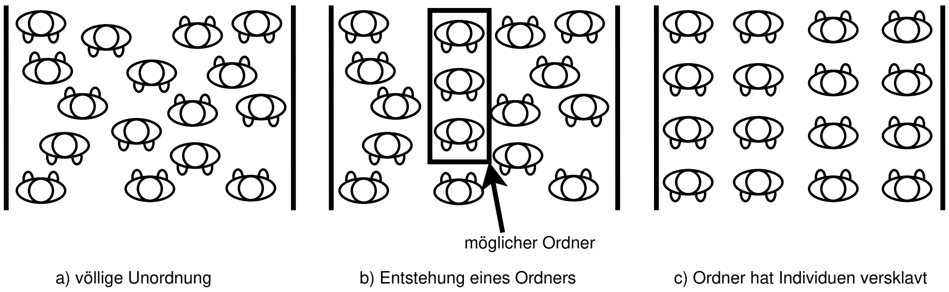


Abbildung 2: Synergetik am Beispiel Entstehung von Ordnung auf einer Treppe

nötig das genaue Verhalten der einzelnen Individuen zu kennen, es reicht zu wissen, welche Ordner für die Individuen maßgebend sind. (Hak91, S. 23) Als Beispiel führt Haken die Erbsubstanz DNS (Desoxyribonukleinsäure) der Lebewesen an. (Hak91, S. 95ff) Trotz des riesigen Umfangs der DNS ist in dieser nicht die Information für jede einzelne Körperzelle abgelegt. Vielmehr enthält die DNS lediglich Informationen für die verschiedenen Zelltypen sowie die Information zur Bildung von Ordnern, die für eine Strukturierung der Zellen sorgen.

Während der Selbstorganisation kann es passieren, dass mehrere Zustände nach dem Phasenübergang gleich wahrscheinlich sind. In dieser Situation entscheidet der Zufall, welcher Zustand sich nach dem Phasenübergang ergibt. (Hak91, S. 109ff) Daraus folgt, dass eine Vorhersagbarkeit nicht möglich ist. Das System neigt zum *Nicht-Determinismus*.

Damit es überhaupt zu einem Phasenübergang kommt, muss dem System Energie zugeführt werden. In sozialen Systemen tritt an die Stelle der Energie die Information. Umgekehrt formuliert bedeutet dies, dass das System Unordnung an seine Umwelt exportiert und von der Umwelt Struktur, in Form von Information, aufnimmt. Durch die Zuführung von Energie (Information) wird der zweite Hauptsatz der Thermodynamik außer Kraft gesetzt. Weiterhin muss das System aus einer größeren Anzahl von Systemelementen bestehen, damit Ordnern entstehen können. Die konkrete Anzahl der benötigten Systemelemente hängt vom System und der *Vernetztheit* der Elemente ab und kann mit dem mathematischen Modell von Haken abgeschätzt werden. Elemente sind dann vernetzt, wenn die Änderung eines Elementes sich über mehrere Stationen auf dieses Element wieder auswirkt. Dies bezeichnet man als *Rückkoppelung*.

Da die Synergetik im Umfeld der Naturwissenschaften entstanden ist, fällt eine Übertragung auf gesellschaftliche Emergenzphänomene schwer und wird von vielen Autoren kritisch hinterfragt. So lassen sich mit der Synergetik z. B. Diktaturen rechtfertigen. Trotzdem unterstützt die Synergetik durch ihre klare sprachliche Formulierung das Verständnis von Emergenz.

## 3.3 Erklärungsmodell Chaostheorie

Haken formuliert, dass die Chaostheorie als eine Untermenge der Synergetik verstanden werden kann. (Hak91, S. 58) Da sich die Chaostheorie jedoch relativ unabhängig von der Synergetik entwickelt hat, erfolgt hier ein kurzer Überblick. Auf eine mathematische Darstellung wird verzichtet und stattdessen eine topologische Darstellung (nach Flä98, S. 127ff) gewählt.

Der Begriff *Chaos* mag irreführend sein, da es sich nicht um die Bedeutung von Chaos im allgemeinen Sprachgebrauch handelt. Die Chaostheorie betrachtet kein völlig zufälliges Verhalten, wie z. B. die Brownsche Bewegung[7]. Um diesen Unterschied hervorzuheben, sprechen manche Autoren von *deterministischem Chaos*[8].

Ausgangspunkt der Chaostheorie waren Untersuchungen durch den Meteorologen Lorenz in der 60er Jahren. Er versuchte die Wetterentwicklung mit einem Gleichungssystem vorherzusagen, welches lediglich auf drei Variablen (Temperatur, Luftdruck und Windrichtung) und drei Gleichungen basiert. Dieser Versuch scheiterte, da schon kleine Messfehler bei der Erhebung der aktuellen Wetterdaten zu sehr unterschiedlichen Vorhersagen führten. Obwohl eine mathematische Gleichung zugrunde lag, entwickelten sich die Ergebnisse unvorhersagbar. In diesem Zusammenhang wurde der Begriff deterministisches Chaos gebildet. Von Lorenz stammt das Sinnbild, wonach ein Schmetterlingsschlag[9] einen Sturm irgendwo anders auf der Welt auslösen kann.

Der Zustand eines Systems bildet in einem beliebig dimensionalen Raum, bezeichnet als *Phasenraum*, einen Punkt. Der Phasenraum kann z. B. durch die Dimensionen Ort, Zeit und Geschwindigkeit gekennzeichnet werden. Zwischen den verschiedenen Zustandspunkten im Phasenraum ergibt sich eine Kurve. Diese Kurve wird als *Trajektorie* bezeichnet. Streben nun alle Trajektorien eines Systems, ausgehend von verschiedenen Startpunkten, auf einen bestimmten Bereich im Phasenraum asymptotisch zu, wird dieser Bereich als *Attraktor* bezeichnet. Der Attraktor bildet im Phasenraum einen Bereich, in dem sich das System relativ stabil verhält. Es ergibt sich das in Abbildung 3 auf Seite 15 dargestellte Diagramm. Interessant ist weiterhin, dass

[7] Bereits im 19. Jahrhundert beobachtete Robert Brown unter dem Mikroskop eine scheinbar völlig zufällige Bewegung von Pollen. Erst gut 100 Jahre später konnte gezeigt werden, dass die Pollen durch viel kleinere „unsichtbare“ Teilchen (z. B. Moleküle) gestoßen werden und sich deshalb die zufällige Bewegung ergibt. Die Bewegung der Pollen (oder anderer Partikel) ist ein stochastischer Prozess und wird als Brownsche Bewegung bezeichnet.

[8] Der Begriff deterministisches Chaos besagt, dass obwohl es einen Algorithmus zur Berechnung gibt, keine Vorhersage getroffen werden kann, ohne den Algorithmus vollständig auszuführen. Dies lässt sich am Beispiel der Zahl $\pi$ nachvollziehen. Es ist unmöglich z. B. die tausendste Stelle von $\pi$ genau vorher zu sagen, ohne dass man die Berechnungsvorschrift für $\pi$ durchführt. (vgl. PJS94, S. 11ff)

[9] Lorenz hat wohl von dem Flügelschlag einer Taube gesprochen, aber der Schmetterlingsschlag hat sich als Metapher durchgesetzt.

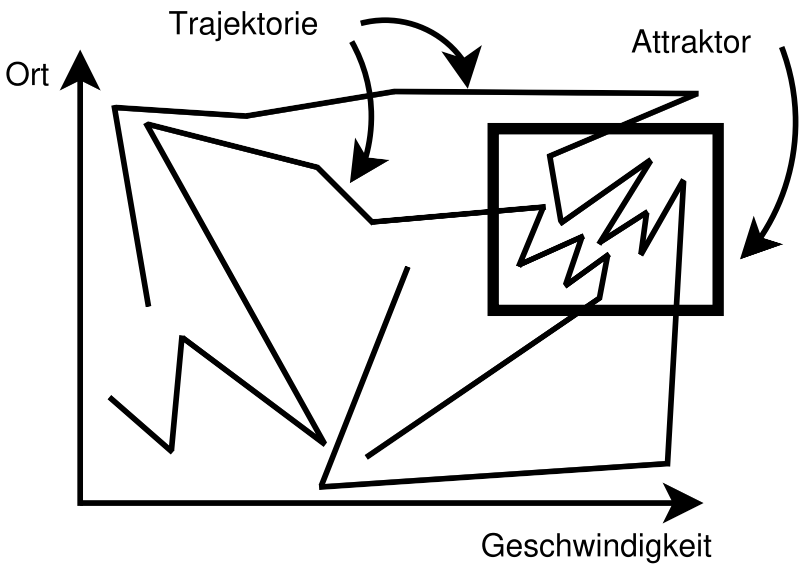


Abbildung 3: zweidimensionaler Phasenraum mit Trajektorien und Attraktor

der Bereich des Attraktors eine niedrigere Dimension als der Phasenraum besitzt und somit eine bessere Untersuchung des Systems möglich ist, da eine Vereinfachung vorliegt. Zum Verständnis soll erwähnt werden, dass im Rahmen der mathematischen Darstellung der Chaostheorie nicht ganzzahlige Dimensionen zulässig sind.

In der Chaostheorie ist ein Attraktor von besonderem Interesse. Dieser wird als *seltsamer Attraktor* bezeichnet und zeigt einige spezielle Eigenschaften. Die sich dem seltsamen Attraktor asymptotisch nähernden Trajektorien erreichen jeden Punkt des seltsamen Attraktors ohne sich gegenseitig zu schneiden. Hier spricht man von *Struktur*. Auf der anderen Seite ist das Verhalten der Trajektorien völlig chaotisch, da die Trajektorien zwischen den einzelnen Punkten des seltsamen Attraktors ohne erkennbare Logik springen. Benachbarte Punkte liegen demnach auf unterschiedlichen Trajektorien und dadurch lässt sich begründen, warum kleinste Änderungen, *Fluktuationen*, zu sehr unterschiedlichen Ergebnissen führen.

Vom Phasenraum zu unterscheiden ist der *Parameterraum*. In diesem Raum wird das Verhalten verschiedener Systeme dargestellt. Die Systeme sind sich ähnlich, weichen lediglich durch die Werte der Parameter ab. Prinzipiell verlaufen die Kurven im Parameterraum kontinuierlich, allerdings kann es passieren, dass sich an einem Punkt das Verhalten des Systems grundlegend ändert und somit eine Änderung der Systemparameter stattfindet. Diese Punkte werden als *Bifurkationspunkte* bezeichnet.

Während der Verlauf im Phasenraum mittels der Trajektorien nachvoll-

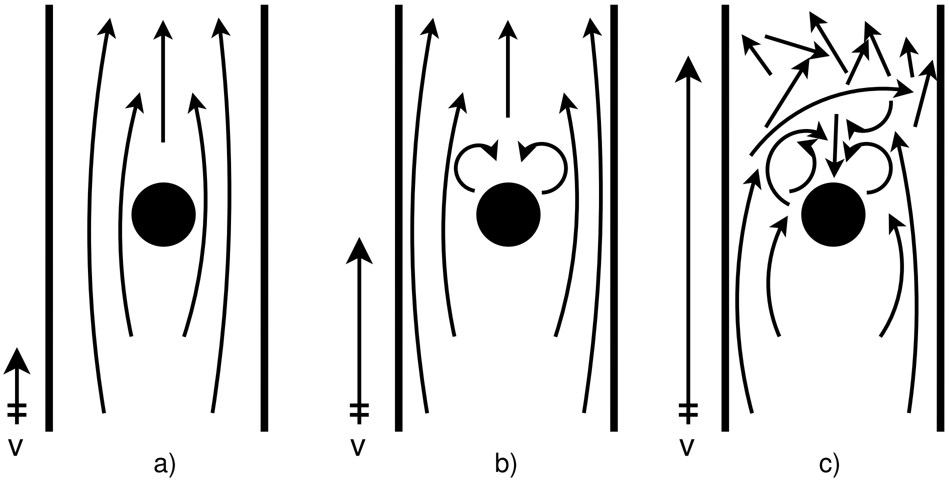


Abbildung 4: Chaostheorie am Beispiel Wasserströmung in Kanal

zogen werden kann, scheint der Verlauf im Parameterraum völlig willkürlich. Deshalb spricht man von deterministischem Chaos. Der Verlauf im Parameterraum kann als Emergenz betrachtet werden. Eine Vorhersage scheint prinzipiell nicht möglich.

An dieser Stelle wird an einem kurzen Beispiel die Chaostheorie verdeutlicht. Das Beispiel geht zurück auf Haken (Hak91, S. 59), wurde aber für diese Arbeit abgewandelt. Man stelle sich einen idealen geradlinigen Wasserkanal vor, wie in Abbildung 4 auf Seite 16 gezeigt. In dessen Mitte befindet sich ein Zylinder, um den das Wasser herumströmt. Bei niedriger Fließgeschwindigkeit wird keine Auffälligkeit zu beobachten sein (Abbildung 4 Teilbild a)). Das Wasser strömt gleichmäßig um den Zylinder. Erhöht man die Fließgeschwindigkeit kontinuierlich, werden sich ab einer bestimmten Geschwindigkeit Wirbel im Wasser hinter dem Zylinder bilden, wie in Abbildung 4 Teilbild b) gezeigt. Die Wirbelbildung ist ein *Attraktor*. Das Umschlagen des Verhaltens des fließenden Wassers stellt eine *Bifurkation* dar. Durch eine weitere Erhöhung der Fließgeschwindigkeit nehmen die Wirbel zu und das Verhalten des Wassers hinter dem Zylinder wird dadurch zunehmend komplizierter zu beschreiben. Bei der weiteren Erhöhung der Fließgeschwindigkeit wird ein weiterer Bifurkationspunkt erreicht. Danach bewegt sich das Wasser hinter dem Zylinder völlig chaotisch (Abbildung 4 Teilbild c)) und es lässt sich kein klares Verhalten, wie z. B. Wirbel, erkennen. Somit streben die *Trajektorien*, die den Zustand des Wassers hinter dem Zylinder beschreiben, auf den *seltsamen Attraktor* zu. In diesem Gedankenexperiment wurde lediglich die Fließgeschwindigkeit des Wassers erhöht. Trotzdem ergab sich daraus völlig unerwartetes (nicht-deterministisches) und spontanes (nicht-lineares) Verhalten. Dies entspricht dem Gedanken der Emergenz.

## 3.4 Erklärungsmodell Autopoiesis

Abschließend wird die Autopoiesis nach Maturana betrachtet. Aus der Autopoiesis entwickelte Luhmann später seine soziologische Systemtheorie (vgl. z. B. Sta94), die hier nicht betrachtet wird, da sie unter heftiger Kritik[10] steht. (z. B. bei Büh87)

Prinzipiell lassen sich zwei Ausgangspunkte (Fis93, S. 9) für die Autopoiesis erkennen. Auf der einen Seite versuchte Maturana, zusammen mit seinen Kollegen (speziell Varela), die nach dem zweiten Weltkrieg entstandene Kybernetik[11] auf die Biologie zu übertragen. Auf der anderen Seite soll die Autopoiesis als ein naturwissenschaftliches Erklärungsmodell der Erkenntnis fungieren. (vgl. Fis93) In diesem Rahmen soll sie klären, wie der Mensch zur Erkenntnis (Wissen) gelangen kann. Nach Maturana sind lebende Systeme stets autopoietisch. Dieser Aspekt der Autopoiesis ist an dieser Stelle nicht von Interesse.

Der Begriff Autopoiesis bedeutet soviel wie „Selbsttun“ oder „Selbstgestaltung“. Damit wird ein wesentlicher Aspekt der Autopoiesis angesprochen. Demnach dürfen nur Systeme als autopoietisch bezeichnet werden, die ihre Systemelemente selbst erzeugen. Alle Systemelemente müssen aus den vorhandenen Systemelementen entstehen. In diesem Zusammenhang spricht man von *Zirkularität*. Es werden keine Systemelemente aus der Umwelt in das System übernommen. Weiterhin müssen autopoietische Systeme abgeschlossen gegenüber der Umwelt sein. Damit ist gemeint, dass eine Strukturveränderung nur aus dem System selbst heraus entstehen kann. Nicht gemeint ist damit eine energetische oder informationelle Abgeschlossenheit gegenüber der Umwelt, denn Strukturänderungen auslösende Systemstörungen können durchaus durch Umwelteinflüsse erfolgen. Man bezeichnet diese Eigenschaft, dass eigene Zustände nur im System intern bestimmt werden, als *Selbstreferentialität*. Das System wählt durch die Festlegung der Systemgrenze den Umfang und die Art des Kontakts zur Umwelt. Diese Eigenschaft wird als *strukturelle Koppelung* bezeichnet und ersetzt die Input/Output Beziehung der Kybernetik. Aufgrund dieser Interpretation der Systemgrenze ist das System nicht fähig Zustandsänderungen der Umwelt wahrzunehmen. Auf der anderen Seite kann ein externer Beobachter keine Aussagen über die interne Organisation des autopoietischen Systems treffen. Man bezeichnet dies als *operative Geschlossenheit*. Von Außen kann lediglich eine Betrachtung der Input/Output Beziehung erfolgen. Welche internen Vorgänge für die Erzeugung eines Outputs bei einem bestimmten Input verantwortlich

[10] Luhmann formulierte seine Systemtheorie für die Soziologie mit dem Anspruch allgemeiner Gültigkeit. Dadurch scheint seine Theorie dem Historizismus (vgl. 2.3 auf Seite 8) sehr nahe. Ob diese Kritik berechtigt ist, kann an dieser Stelle nicht beurteilt werden.

[11] Hier ist die ursprüngliche Kybernetik nach Wiener (vgl. Wie71) in den 40er und 50er Jahren gemeint und nicht die später etablierte Kybernetik höherer Ordnung z. B. nach von Foerster.

sind, ist nicht analysierbar. Das System entspricht damit einer Blackbox[12].

Durch die operative Geschlossenheit und Selbstreferentialität autopoietischer Systeme ist eine gezielte Beeinflussung des Systems unmöglich. Da die Umwelt den Zustand des autopoietischen Systems nicht erkennen kann, kann die Umwelt nicht beurteilen, wie das System auf einen Umwelteinfluss, eine Störung, reagiert. Obwohl das System intern völlig deterministisch abläuft, verhält es sich nach Außen (scheinbar) *nicht-deterministisch*. (Flä98, S. 169ff)

Ein autopoietisches System kann deshalb als „*selbstorganisierend*, *selbsterzeugend*, *selbsterhaltend* und *selbstreferentiell*" (vgl. Flä98, S. 163)[13] beschrieben werden, obwohl mit der Kybernetik eine mechanische Sichtweise[14] die Basis bildete. Man spricht in der Autopoiesis von Selbstorganisation, da das autopoietische System spontan seinen eigenen Zustand an Randbedingungen (strukturelle Koppelung) anpassen kann.

Selbst die Autopoiesis nach Maturana steht unter Kritik, da das grundlegende biologische Experiment[15] niemals vollständig nachvollzogen werden konnte. Auch erscheint die Möglichkeit von *Selbstreferentialität* bei gleichzeitiger *struktureller Koppelung* als eine willkürliche Festlegung. Dennoch hat die Autopoiesis im Bereich der Biologie und der Soziologie den Gedanken der Selbstorganisation etabliert. Eine Vielzahl von Managementpraktiken wurde von der Autopoiesis inspiriert.[16]

Der Bezug zur Emergenz ergibt sich, wenn man betrachtet, dass in einem autopoietischen System durch Selbsterzeugung und Selbstreferenz eine Vielzahl von Systemelementen organisiert werden und dabei in ihrer Gesamtheit höhere Eigenschaften hervorbringen. In der Theorie der Autopoiesis wird betont, dass in einem autopoietischen System neben den Systemelementen eine systemspezifische Organisation herrscht. Dabei geht man davon aus, dass einzelne Systemelemente austauschbar sind, solange die spezifische Organisation erhalten bleibt. Darin zeigt sich, dass das Systemverhalten nicht auf das Verhalten der einzelnen Elemente zurückgeführt wird, sondern dass neben den Systemelementen eine spezifische Organisation entsteht, die für das

[12] Eine Blackbox bedeutet, dass die Analyse der im System ablaufenden Vorgänge nicht möglich ist. Somit kann eine Untersuchung des Systems lediglich anhand der Eingaben und Ausgaben erfolgen.

[13] Hervorhebungen durch Verfasser

[14] zum mechanischen Weltbild siehe Kapitel 4.2 auf Seite 21

[15] Maturana hat für dieses Experiment Elektroden in die Ganglienzellen der Netzhaut von Tauben eingepflanzt. Er zeigte den Tauben unterschiedliche Farbtafeln. Es wurde erwartet, dass die unterschiedlichen Farben zu einer jeweils eigenen Stimulation der Ganglienzellen führen würden. Dem war aber nicht so. Daraus schloss Maturana in späteren Arbeiten, dass das Nervensystem nicht von äußeren Reizen gesteuert wird, sondern diese Reize lediglich Auslöser von Veränderungen im Nervensystem sind. Diese Gedanken mündeten letztendlich in der Autopoiesis.

[16] Dies kann man durch die Suche nach den Begriffen Autopoiesis und Management im Internet nachvollziehen. In Kapitel 5.2 auf Seite 25 werden einige Anwendungen kurz vorgestellt.

Systemverhalten genauso entscheidend ist. (Flä98, S. 173ff) Man kann deshalb davon ausgehen, dass autopoietische Systeme emergente Eigenschaften zeigen.

## 3.5 Zusammenfassende Vereinheitlichung

Die in diesem Abschnitt vorgestellten Theorien sind eine kleine Auswahl von verfügbaren Erklärungsmodellen für Emergenz. Dennoch kann man bei allen Theorien Parallelen erkennen. Chaostheorie und Synergetik scheinen sehr nah verwandt, eine Zuordnung der Begriffe Ordner und Attraktor sowie Phasenübergang und Bifurkation erscheint prinzipiell möglich.

Flämig (Flä98, S. 56ff) reduziert die Theorien auf *Nicht-Linearität* und *Nicht-Determinismus*. Die Nutzung des Begriffs *Selbstorganisation* lehnt er ab, (vgl. Flä98, S. 61) da durch eine zu breite Nutzung des Begriffs dieser an Aussagekraft verloren hat. (vgl. auch Kra96) Trotzdem wird in dieser Arbeit der Begriff Selbstorganisation weiterhin benutzt als begriffliche Zusammenfassung für Anpassung bzw. Strukturänderung durch kollektive Effekte, da im Gegensatz zu den Begriffen Nicht-Linearität und Nicht-Determinismus der Begriff Selbstorganisation weniger abstrakt ist. Weiterhin erfolgt in dieser Arbeit eine Übernahme und Anwendung der von Flämig vorgeschlagenen Begriffe Nicht-Linearität und Nicht-Determinismus.

Unter *Nicht-Linearität* wird in dieser Arbeit verstanden, dass trotz intensiver Analyse (noch) keine schlüssige Erklärung für die Abfolge von Systemzuständen gefunden werden kann. Das System scheint wahllos zwischen teils entgegengesetzten Systemzuständen zu springen. Bereits kleinste Änderungen im Anfangszustand können vollkommen verschiedene Endzustände hervorrufen.

Unter *Nicht-Determinismus* wird verstanden, dass trotz genauester Messung des Systemzustandes und der Kenntnis aller wirkenden Systemgesetze eine Vorhersage des Systemverhaltens über einen längeren Zeitraum unmöglich ist. Die Kenntnis aller Einzelschritte zwischen Anfangszustand und Folgezustand reicht nicht aus, um die Transformation vollständig zu beschreiben.

Als Grund für Nicht-Linearität wird die Vernetzung, und damit die Ermöglichung von Rückkoppelung, gesehen. Eine mathematische Beschreibung, z. B. mithilfe der Synergetik, führt zu nicht-linearen mathematischen Modellen. Der Nicht-Determinismus ist prinzipiell darauf zurückzuführen, dass der Eintritt mancher Systemzustände auf Zufall beruht. Zufall ist per Definition nicht vorhersagbar und somit kein darauf aufbauendes System.

In einem System, welches Nicht-Linearität und Nicht-Determinismus zulässt, kann Selbstorganisation auftreten. Um Emergenz in der Softwareentwicklung umsetzen zu können, muss Nicht-Linearität und Nicht-Determinismus in der Softwareentwicklung ermöglicht werden. Dazu ist es sinnvoll zu untersuchen, wie bereits in anderen Bereichen diese Forderung umgesetzt wird, um dann die Anwendung im Bereich Softwareentwicklung genauer studieren zu können. Da die Begriffe Nicht-Linearität und Nicht-Determinismus so fundamental sind, wird zunächst betrachtet, welches Bild bei einer aus nicht-linearen nicht-deterministischen Systemen zusammengesetzten Welt sich ergibt. Deshalb wird im nächsten Abschnitt untersucht, wie es zur Entwicklung eines Weltbildes basierend auf Nicht-Linearität und Nicht-Determinismus kam.

# 4 Geschichtliche Entwicklung

## 4.1 Einleitung

Begriffe wie Chaostheorie, Synergetik, Autopoiesis, Selbstorganisation und Emergenz erfreuen sich heutzutage großer Beliebtheit. Allerdings haben sich diese Begriffe und die dahinter stehenden Ansichten erst in den letzten 50 bis 70 Jahren entwickelt. Dabei sprechen eine Vielzahl von Autoren (vgl. z. B. KK92; Flä98; Hei86; PJS94) von einer Ablösung des *mechanischen Weltbildes* nach Newton. In diesem Abschnitt wird zuerst das mechanische Weltbild dargestellt und anschließend das sich neu entwickelnde Weltbild nachgezeichnet. Besondere Beachtung findet dabei die Entwicklung der fundamentalen Begriffe Nicht-Linearität und Nicht-Determinismus.

Abschließend findet eine Systematisierung statt, die die Aufgabenstellung dieser Arbeit anhand des bereits dargestellten Wissens präzisiert.

## 4.2 Mechanisches Weltbild

Mit seinem Werk „Philosophiae naturalis principia mathematica“ legte Isaac Newton im Jahr 1687 den Grundstein des *mechanischen Weltbildes*. Aus den Keplerschen Gesetzen leitete er das Gravitationsgesetz ab, mit dem die theoretische Basis für die Himmelsbewegungen gefunden war. Mit seiner Vereinheitlichung können die grundlegenden physikalischen Begriffe *Masse*, *Impuls* und *Kraft* beschrieben werden. Weiterhin ist die Beschreibung von Schwingungen durch dieses Werk möglich geworden.

Die Newton-Mechanik hat seitdem einen enormen Einfluss auf alle anderen Wissenschaftsdisziplinen. Grundgedanke ist die Maschine, deren Verhalten genau bestimmbar ist. Verfügt man über die Kenntnis des genauen Zustandes der Maschine zu einem Zeitpunkt und den Regeln des Maschinenverhaltens, kann daraus jeder Zustand in der Zukunft bzw. Vergangenheit bestimmt werden. Da sich die Bewegung der Planeten und Sterne mit der Newton-Mechanik genau beschreiben lässt, erscheint es durchaus sinnvoll, die Gesetze auf den Mikrokosmos zu übertragen und die Bewegung von Kernteilchen mit den Gesetzen der Mechanik zu beschreiben.

Die Newton-Mechanik wurde auf viele Bereiche übertragen. So stellte beispielsweise Julien Offray de La Mettrie in seinem Werk „L'homme machine“ den Menschen als eine Maschine dar. (vgl. PJS94, S. 3)[17] Diese Abstraktion hat viele Erkenntnisse in der Medizin ermöglicht, allerdings besteht dadurch auch die Gefahr einer Entmenschlichung der Medizin.

Am Anfang des 20. Jahrhunderts entwickelte der amerikanische Ingenieur und Unternehmer F. W. Taylor (vgl. Sta99, S. 23ff) eine Theorie zur Betriebsführung. Der so genannte *Taylorismus* sieht genaue Arbeitsbe-

[17] Beitrag „Philosophische Reflexionen über Chaos und Ordnung“ von Bernulf Kanitschneider

schreibungen und Zeitvorgaben für die Verrichtung von Arbeitstätigkeiten vor. Der Mensch wird in diesem Arbeitssystem zu einem „Zahnrad“ in einer riesigen Fertigungsmaschine. Fällt das „Zahnrad“ aus, kann es durch einen anderen Menschen ersetzt werden. Dies führte zum Teil zu einer Entmenschlichung der Arbeit, wie Fritz Lang in seinem Film „Metropolis“ zeigt. Andererseits ermöglichten die klaren Aufgabenbeschreibungen, dass selbst ungelernte Kräfte die Tätigkeiten ausführen konnten. Weiterhin kann die Arbeit von Taylor als die Begründung der Arbeitswissenschaften angesehen werden.

Eine weitere Anwendung fand die Newton-Mechanik in der Kybernetik nach Wiener (vgl. Wie71). In der Kybernetik wird Systemverhalten untersucht und anhand von Regelkreisen mit positiver oder negativer Rückkoppelung beschrieben. Die Kybernetik entwickelte sich zur Basis einer umfassenden Steuerungstechnik und war letztlich eine Grundlage der entstehenden Rechentechnik nach dem zweiten Weltkrieg. Auch in der Kybernetik finden sich die Grundideen der Newton-Mechanik wieder. Ein System ist durch seinen Zustand und seine Veränderungsgesetze vollständig beschrieben und es kann prinzipiell das zukünftige Verhalten bei der Kenntnis des System vorhergesagt werden (Determinismus). Weiterhin führen kleine Änderungen der Eingangswerte zu kleinen Änderungen der Ausgangswerte (Linearität), was eine gezielte Steuerung des Systems ermöglicht.

## 4.3 Das neue Weltbild

Während das alte Weltbild konkret benannt werden kann, ist dies für das neue Weltbild nicht möglich, da es sich noch in der Entstehung befindet. Eine Reihe von Entwicklungen haben zu einer Änderung der grundlegenden Ansichten geführt, es ist allerdings schwer den Auslöser auf eine einzelne Person oder eine einzelne Theorie zu reduzieren.

Bereits bei Kant finden sich Anzeichen des Selbstorganisationsgedanken. (vgl. KK92) So akzeptierte Kant durchaus die Idee, dass die Entstehung des Kosmos durch die Newton-Mechanik beschreibbar ist, für die Entstehung von Leben hingegen konnte er dieser Idee nicht zustimmen.

Im 19. Jahrhundert deutete sich z. B. bei den Arbeiten von Boltzmann und Maxwell an, dass die Newton-Mechanik nicht allgemeingültig war. Allerdings wurde die Newton-Mechanik zu diesem Zeitpunkt noch nicht in Frage gestellt. (PJS94, S. 5) Albert Einstein gelang mit seiner Allgemeinen Relativitätstheorie[18] am Anfang des 20. Jahrhunderts eine Darstellung der Begriffe Zeit und Raum. Weiterhin konnte durch die Allgemeinen Relativitätstheorie die Newton-Mechanik auf den Makrokosmos übertragen werden. Wiederum

[18] Die Allgemeine Relativitätstheorie kann als bewiesen betrachtet werden. So stellt sie z. B. die These auf, dass große Massen, wie die Sonne unseres Sonnensystems, Licht ablenken. Dies konnte mehrfach bei totalen Sonnenfinsternissen experimentell nachgewiesen werden.

schien die Newton-Mechanik prinzipiell bestätigt.

Max Planck legte im Jahr 1900 mit der Quantenhypothese[19] die Grundlage der Quantentheorie. Nach der Quantenhypothese kann Energie nicht in beliebig kleine Mengen zerlegt werden. (Flä98, S. 90) Einige Jahre später bauten Nils Bohr und insbesondere Werner Heisenberg (Hei86) die Theorie weiter aus zur Quantenmechanik. Dabei formulierten sie die Unschärferelation, wonach Ort und Impuls eines Elementarteilchens niemals gleichzeitig genau bestimmt werden können. Die Unschärferelation konnte experimentell bestätigt werden. Wenn allerdings keine genaue Messung von Ort und Impuls möglich ist, kann der aktuelle Zustand eines Systems nicht umfassend bestimmt werden. Von dieser unvollständigen Datenlage kann dann allerdings ebenfalls die Zukunft des Systems nicht genau vorherbestimmt werden. Hier zeigt sich deutlich die Abkehr von der Newton-Mechanik. Dieses Problem lässt sich z. B. am Zerfall radioaktiver Teilchen nachvollziehen. Für eine große Menge radioaktiven Materials kann eine Halbwertszeit für den radioaktiven Zerfall angegeben werden. Es ist allerdings unmöglich, genau vorherzusagen, wann ein einzelnes radioaktives Teilchen zerfallen wird. Selbst Einstein akzeptierte die Vorstellung von Zufall als Basis der Quantenmechanik nicht, was sich an seinem Ausspruch „Gott würfelt nicht!“ zeigt. Man kann sagen, Einstein relativierte die Begriffe Zeit und Raum, Heisenberg ging einen Schritt weiter und relativierte den Begriff Kausalität (Ursache und Wirkung).

Eine Reihe von Wissenschaftlern suchte nach Gründen, warum sich die Welt zu stets komplexeren Ordnungen hin entwickelt, obwohl dies dem zweiten Hauptsatz der Thermodynamik widerspricht. Würden lediglich die physikalischen Naturgesetze wirken, würde im Laufe der Zeit jegliche Energie und Materie im Universum verbraucht und das Universum würde dem „Wärmetod“ zusteuern. In diesem Zusammenhang entstanden dann ab den 60er Jahren die bereits vorgestellten Erklärungsmodelle für Selbstorganisation, wie Synergetik und Autopoiesis.

Das neue Weltbild befindet sich noch immer in der Entstehung und es ist noch nicht absehbar, wie es weitergestaltet wird. Dennoch finden die Ideen von Selbstorganisation, Nicht-Determinismus und Nicht-Linearität bereits heute ihre Anwendung, was im folgenden Kapitel genauer gezeigt wird.

[19] Die Quantenhypothese besagt, dass Licht nicht in beliebig großen Energieportionen abgegeben werden kann, sondern dass die Energieportion immer ein ganzzahliges Vielfaches des Energiequants ist. Das Energiequant ist das Produkt von Lichtfrequenz und einer Naturkonstante (später als Plancksches Wirkungsquantum bezeichnet). Aufbauend auf dieser Hypothese entstand die Quantentheorie. Die Quantentheorie sagt aus, dass Vorgänge in der Natur auf mikroskopischer Ebene (auf Ebene der Elementarteilchen) nicht kontinuierlich, sondern sprunghaft ablaufen. Weiterhin besagt die Quantentheorie, dass Vorgänge nicht genau vorhersagbar sind, sondern lediglich eine Aussage über die Wahrscheinlichkeit des Eintretens bestimmter Ereignisse getroffen werden kann. Die Quantenmechanik untermauert dies und sagt, dass über Ort, Impuls, Energie und Zeit eines Teilchens keine unendlich genauen Aussagen getroffen werden können.

## 4.4 Systematisierung

Nach der Akzeptanz des im vorherigen Unterabschnitt vorgestellten neuen Weltbildes ergibt sich die Aufgabenstellung, Mittel und Wege für die Orientierung in dieser spontanen und nicht vorherbestimmbaren Welt zu finden. Dies ist die erste sich ergebende Aufgabenstellung – eine Reaktion auf das herrschende Weltbild und somit eine Bewältigung von Nicht-Linearität und Nicht-Determinismus.

Neben der Reaktion gibt es die Aktion, also die bewusste Gestaltung eines dem Weltbild entsprechenden Systems. Man muss fragen, wie man ein emergentes System gestalten und nutzen kann. Es geht somit um die Erzeugung von Nicht-Linearität und Nicht-Determinismus zur Lösung von Problemen. Dies ist die zweite Fragestellung.

Die Kombination beider Fragestellung führt zu einem „proaktiven“[20] Ansatz. Dieser Ansatz liefert Mittel und Wege für

1. die *Bewältigung* der Anforderungen gestellt durch eine emergente Welt

2. und für die *Gestaltung* einer emergenten Welt.

Während die Bewältigung einer emergenten Welt lediglich eine notwendige Bedingung darstellt, ist die Gestaltung einer emergenten Welt eine Chance. Jeder, der in der emergenten Welt „überleben“ will, muss mit Emergenz umgehen können. Aber derjenige, der selbst Emergenz erzeugen und gestalten kann, kontrolliert diese Welt. Ziel muss deshalb die Gestaltung und Nutzung von Emergenz sein.

Im nun folgenden Abschnitt wird untersucht, wie das neue Weltbild in der Wirtschaftsinformatik bewusst oder unbewusst zugrunde gelegt wird. Am Ende findet eine Zuordnung der einzelnen vorgestellten Verfahren und Theorien zu den beiden Kategorien *Bewältigung* und *Gestaltung* statt.

[20]Der Begriff proaktiv taucht später bei der Untersuchung von Agenten im Abschnitt 5.3.1 auf Seite 31 nochmals auf.

# 5 Anwendung von Emergenz

## 5.1 Einleitende Worte

In diesem Abschnitt wird gezeigt, wie bereits heute eine Anwendung von Emergenz in der Wirtschaftsinformatik stattfindet. Dabei wurden Anwendungen aufgenommen, die zumindest ein identifiziertes Merkmal (Nicht-Linearität, Nicht-Determinismus, Selbstorganisation) aufweisen. Der hier dargestellte Überblick erhebt keinen Anspruch auf Vollständigkeit, sondern vermittelt lediglich ein Bild, wie Emergenz bereits praktisch umgesetzt wird.

## 5.2 Teilbereich Wirtschaft

### 5.2.1 Managementtheorien

In der Managementtheorie ist allgemein ein Wandel festzustellen. Es wird von einer „steigenden Komplexität, Diskontinuität und Dynamik“ (Wei01, S. 255) als Kernmerkmale der neuen Unternehmensumwelt gesprochen, wobei die Überlebensfähigkeit des Unternehmens von seiner permanenten Fähigkeit zur Weiterentwicklung abhängt, um sich der Umwelt anzupassen und diese aktiv zu gestalten. (vgl. Wei01, S. 255)

Diese Sicht kann mit einer Bevorzugung des Konstruktivismus gegenüber dem (kritischen) Rationalismus begründet werden. Der Rationalismus geht prinzipiell davon aus, dass es dem Menschen möglich ist, die Welt zu erkennen und zwischen Wahrheit und Unwahrheit zu unterscheiden. Daraus können Theorien abgeleitet werden, die dann zu falsifizieren sind. (vgl. Wei01, S. 3ff) Der Konstruktivismus geht hingegen davon aus, dass jegliche Erkenntnis über die Umwelt lediglich eine Konstruktion des Menschen ist – der Mensch konstruiert Wahrheiten. Eine objektive Überprüfung von Wahrheiten ist nicht möglich, da wir dazu die Umwelt, wahrgenommen über die menschlichen Sinne, mit der Theorie vergleichen müssen. Es konnte in biologischen Experimenten[21] gezeigt werden, dass Wahrnehmungen stets einer Verarbeitung durch das menschliche Gehirn unterliegen und somit subjektiv geprägt sind. Aus diesem Grund lässt sich keine objektive Wahrheit ableiten. (vgl. Wei01, S. 31ff) Trotzdem ist es nach Meinung der Konstruktivisten sinnvoll, Theorien aufzustellen. Eine Theorie ist dann von hoher Qualität, wenn sie sich in der praktischen Anwendung allgemein bewährt. Das bedeutet aber auch, dass Theorien immer nur unter bestimmten Randbedingungen gültig sind und somit für unterschiedliche Randbedingungen verschiedene Theorien existieren müssen. Dadurch entsteht eine Vielzahl von Theorien und die Existenz einer allgemeinen in jeder Situation gültigen Theorie wird verneint. Somit ist eine Vorhersage von Verhalten aufgrund einer Theorie nur unter engen Randbedingungen möglich, da schon leicht

[21] Es gab eine Reihe von Experimenten, wie z. B. das in Kapitel 3.4 auf Seite 17 in einer Fußnote beschriebene Experiment von Maturana.

geänderte Randbedingungen eine eventuell neue Theorie erfordern (Nicht-Linearität).

Für die Praxis bedeutet dies, dass sich z. B. das Unternehmen ständig auf eine wechselnde Umwelt einstellen muss, indem es fortwährend seine Prozesse (seine internen Theorien) anpasst. Zur Bewältigung dieses Problems kann die Theorie der *lernenden Organisation* genutzt werden. Der eigentliche Lernprozess findet auf der Mikroebene statt. Die Mitarbeiter erkennen und bewältigen Probleme während der täglichen Arbeit. Aus der Sicht des Unternehmens ist nun sicherzustellen, dass diese neuen Erkenntnisse in das Wissen der Gesamtorganisation eingehen. Es sollen somit kleine Änderungen (Fluktuationen) im Wissen der Mitarbeiter zu einer Veränderung auf der Makroebene der Organisation führen. „Es ist nicht die Organisation, die bei der Betreuung von Projekten lernt. Vielmehr erweitern einzelne Mitarbeiter (. . . ) ihren Wissensvorrat“. (Wei01, S. 255) Dies entspricht dem Gedanken der Emergenz. In diesem Zusammenhang kommt der Verwaltung des vorhandenen und gewonnenen Wissens im Rahmen des *Wissensmanagement* eine hohe Bedeutung zu. (vgl. GF03) Zahlreiche Autoren stellen dabei den Mitarbeiter als Wissensträger in den Mittelpunkt ihrer Betrachtungen. (z. B. GF03, S. 167f) Indirekt vollzieht sich damit die Entwicklung hin zu einer menschlicheren Arbeitsperspektive im Gegensatz zum Taylorismus. (vgl. Sta99, S. 23ff) Dabei wird das Individuum als entscheidender Erfolgsfaktor für das Unternehmen begriffen.

Einen weiteren Lösungsvorschlag für ein dynamisches und sich diskontinuierlich veränderndes Unternehmensumfeld ist das *virtuelle Unternehmen*. Ein virtuelles Unternehmen ist der lose Zusammenschluss von verschiedenen Leistungserbringern, wobei für den Kunden dieser Zusammenschluss als Einheit auftritt. Diese Zusammenschlüsse sind meist kurz- oder mittelfristig ausgelegt sowie inhaltlich begrenzt. Zwischen den Partnern des virtuellen Unternehmens besteht kein juristischer Zusammenschluss im Sinne einer juristischen Gesellschaft. Dabei geht ein Unternehmen mit einer Vielzahl von anderen Unternehmen eine Kooperation ein, es entsteht ein virtuelles Unternehmensnetzwerk. Dem virtuellen Unternehmen ist es dadurch möglich, verschiedenste Kundenwünsche durch Rückgriff auf die Ressourcen im Unternehmensnetzwerk zu realisieren und somit im dynamischen Markt zu bestehen. Auch hier steht wiederum die Vorstellung, dass durch ein Kooperationsnetzwerk von Einzelindividuen eine Gesamtleistung hervorgebracht werden kann, die sich nicht auf die Einzelleistungen reduzieren lässt. (vgl. z. B. Bul96, S. 52ff)

Neben einer Vernetzung von externen Spezialanbietern im virtuellen Unternehmen, kann dieses Prinzip ebenfalls auf ein Unternehmen selber angewendet werden. Dazu bildet man im Unternehmen eine Vielzahl von Spezialteams mit großen Handlungsspielräumen, die jeweils auf wenige Aufgabenbereiche fokussieren. Zur Bewältigung eines komplexen Problems werden mehrere Spezialteams miteinander vernetzt. Dieser Ansatz wird als *fraktales*

*Unternehmen* bezeichnet. (vgl. z. B. Bul96, S. 49ff) Der Begriff Fraktal steht dabei für die Selbstähnlichkeit, „d. h. jedes Bruchstück (Fraktal) eines *Ganzen* enthält wiederum die Gesamtstruktur des Ganzen". (Bul96, S. 49)[22] Dabei wird bewusst auf eine Selbstorganisation der Fraktale gesetzt (vgl. War02, S. 270ff), indem sich die einzelnen Fraktale zur Lösung des komplexen Problems selbständig gruppieren sollen. (Bul96, S. 50)

Ein weiteres Konzept, das Ende der 80er Jahre in Erscheinung trat (vgl. WJR94), ist die schlanke Produktion (Lean Production). Dieses Konzept wurde später zum schlanken Management (Lean Management) weiterentwickelt. Die Ursprünge liegen in der Untersuchung der weltweiten Automobilproduktion und den teilweise gravierenden Produktivitätsunterschieden zwischen den einzelnen Produzenten. In der Untersuchung wurde festgestellt, dass die japanischen Automobilhersteller wesentlich effizienter produzieren, indem sie sich von einer reinen Massenfertigung abgewandt haben. Tatsächlich fand eine Kombination aus Massenfertigung und Individualfertigung statt, um die Vorzüge beider Produktionsmethoden zu nutzen. Im Lean Production findet eine Verlagerung der Verantwortung hin zu denen statt, die die eigentliche Wertschöpfung erbringen. In der Automobilproduktion sind dies die Techniker am Fließband bzw. den Fertigungsstationen. Damit diese Mitarbeiter ihrer Verantwortung gerecht werden können, findet ein aktiver Ausbau der Mitarbeiterfähigkeiten, z. B. durch Ausbildung, statt. Die Arbeit erfolgt in Teams. Innerhalb der Teams findet eine weitgehende Selbstorganisation in Hinblick auf die zu erledigende Arbeitsaufgabe statt. Zur Absicherung wird ein dichtes Netz aus Qualitätskontrollen und „proaktiven Qualitätszirkeln"[23] gebildet. Die Teams arbeiten möglichst auf engem Raum, damit alle Teams Probleme bei anderen Teams bemerken und bei der Lösung aktiv unterstützen können. Weiterhin wird der Kunde partnerschaftlich in den Produktionsprozess mit einbezogen. Erst auf seinen Auftrag hin findet die Fertigung statt. Das Management sorgt für die nötigen Rahmenbedingungen, es greift selten direkt ein. Die mittleren Führungsebenen können wesentlich reduziert werden, da die Entscheidungen direkt auf den unteren Ebenen getroffen werden. Im Lean Production bzw. Lean Management wird somit versucht, Selbstorganisation zu initiieren und anzuwenden.

Alle genannten neuen Betrachtungsweisen zur Führung und Gestaltung eines Unternehmens können mit den Begriffen *agil*, *antizipativ* und *adaptiv* (nach Mil02, S. 9f) zusammengefasst werden. Dabei versteht man unter Agilität „Schnelligkeit, Beweglichkeit und Flexibilität - für kurze Reaktionszeiten auch bei unerwarteten Hindernissen." (Mil02, S. 9)[24] Unter antizipativ wird das Erkennen der aktuellen Situation und deren wahrscheinliche Weiterentwicklung verstanden. Dies bedeutet eine Abkehr von umfangreicher

[22] Hervorhebungen im Original

[23] In diesen Zirkeln versuchen die Mitarbeiter aus begangenen Fehlern zu lernen, aber auch zukünftige Fehler zu vermeiden.

[24] Der Begriff Agilität wird im Kapitel 7.5 auf Seite 63 nochmals ausführlich aufgegriffen.

Planung hin zu einer Beurteilung der nötigen Mittel anhand der aktuellen Situation. (vgl. Mil02, S. 10)

Die mit dem Begriff adaptiv angesprochene Anpassungsfähigkeit des Unternehmens an eine sich ändernde Umwelt, kann prinzipiell über zwei Maßnahmen erfolgen. Zum einen kann sich die interne Struktur des Unternehmen ändern, eine Änderung nach Außen ist nicht sichtbar. Dem entgegengesetzt ist die Änderung der Schnittstellen des Unternehmen zur Umwelt unter Beibehaltung der inneren Struktur des Unternehmens. Die Stabilität beider Bereiche während einer Anpassung ist nicht möglich. Deshalb kann nur eine Anpassung durch die Änderung der inneren Struktur erfolgen, da das Unternehmen nach Außen stabile Schnittstellen anbieten sollte. Um eine ständige Anpassung, beispielsweise realisiert über die weiter oben diskutierte lernende Organisation, des Unternehmens zu ermöglichen, bedarf es deshalb der Lernfähigkeit aber auch der Lernbereitschaft im Unternehmen.

Zusammenfassend kann gesagt werden, dass eine Vielzahl von neuen Ideen in der Managementlehre entstanden ist. Den meisten liegt eine Abkehr von dem mechanischen Weltbild nach Newton zugrunde, hin zu einer Sichtweise zwischen Nicht-Linearität, Nicht-Determinismus und Selbstorganisation. Dies ist im Ansatz Umsetzung von Emergenz.

### 5.2.2 Theorie steigender Erträge

Die Betriebswirtschaftslehre geht davon aus, dass ein Produzent im Laufe der Zeit mit sinkenden Erträgen rechnen muss. Dies wird begründet durch eine Zunahme der Konkurrenz auf dem Markt, den der Produzent bedient. Weiterhin bestehen natürliche Beschränkungen, wie Rohstoffe, Boden und die Anzahl der Kunden. Durch diese Randbedingungen soll es keinem Produzenten möglich sein, sich langfristig in einem Markt festzusetzen und diesen zu dominieren, da ein neuer Konkurrent ihm die Marktposition immer wieder streitig machen wird.

Die Realität zeigt, dass dies nicht immer so ist. In der Weltwirtschaft haben sich eine Reihe von Großkonzernen gebildet, die in bestimmten Teilbereichen, wie z. B. bei Standardsoftwaresystemen, ein Quasi-Monopol innehaben.

Der amerikanische Wissenschaftler W. Brian Arthur entwickelte die Theorie der steigenden Erträge zur Klärung dieser Phänomene. (vgl. Art96) Er geht davon aus, dass im Bereich der Verarbeitung nichtmaterieller Güter die Gesetze der sinkenden Erträge nicht zwangsläufig gelten. Er bezieht seine Theorie speziell auf die informationsverarbeitende Wirtschaft. Dazu zählt nicht die Bewirtschaftung materieller Güter und die Bereitstellung von Dienstleistungen. Erreicht ein Anbieter in solch einem Markt die Vorherrschaft, kann diese nicht durch Konkurrenten einfach gebrochen werden. Aus der Vorherrschaft ergeben sich Vorteile, die die Marktposition des Anbieters sichern und weiter ausbauen. So kann der Anbieter z. B. eigene Standards vorschreiben oder durch Bündelung von verschiedenen Produkten für eine Migration der Kunden zwischen den Produkten sorgen. Weiterhin entstehen eine Vielzahl von kleinen Anbietern, die die zugrunde liegende Technologie des Marktführers stützen, da sie Synergieeffekte erwarten. Dadurch wird die vom Anbieter bereitgestellte Technologie attraktiver und zieht wiederum mehr Kunden an. Es entsteht eine positive Rückkoppelung im Sinne des Regelkreises. Durch die Beherrschung des Marktes durch den Anbieter kann er mit steigenden Erträgen rechnen, da im Bereich der Informationswirtschaft die Produktionskosten meist gegen Null streben und somit durch jede zusätzlich abgesetzte Einheit die Entwicklungskosten refinanziert werden können.

Arthur betont (Art96, S. 103f), dass die Marktbeherrschung des Anbieters durch neue Technologien gebrochen werden kann. Jeder Anbieter ist deshalb bestrebt, die nächste Technologiewelle frühzeitig zu identifizieren und möglichst zu besetzen. Allerdings hat der Marktführer den Vorteil, dass er einen entscheidenden Einfluss darauf hat, welche Technologie sich durchsetzen wird. Er kann die Technologie bei seinen Kunden einführen und somit bereits für eine breite Anwenderbasis sorgen, was den Erfolg der Technologie wesentlich erhöht.

Als Beispiel führt Arthur (Art96, S. 102) die Entwicklung auf dem Markt der Computer Betriebssysteme in den 80er Jahren an. Zu dieser Zeit gab es drei konkurrierende Technologien (CP/M, MS DOS, Mac OS). Bekanntermaßen setzte sich MS DOS durch, da es von IBM als Betriebssystem für den PC gewählt wurde. Da die IBM-Plattform frühzeitig einen großen Kundenkreis erreichte und dadurch sehr attraktiv für Entwickler wurde, konnte sich MS DOS durchsetzen, obwohl es technisch nicht die Beste der drei Technologien war. Später nutzte Microsoft, der Hersteller von MS DOS, seine Marktmacht, um seine Kunden auf MS Windows und MS Office zu migrieren. Durch die riesigen Absatzmengen von MS DOS und späteren Produkten, konnte Microsoft seine Entwicklungskosten unter einer Vielzahl von Kunden aufteilen und das gewonnene Kapital zur Entwicklung und Durchsetzung weiterer Technologien nutzen.

Die Theorie der steigenden Erträge hat aus Sicht der Informatik einen besonderen Reiz. Sie beachtet die besonderen Eigenschaften des Produktes und Produktionsmittels *Information*. Dadurch erscheint sie ein besseres Erklärungsmodell zu liefern, als die Übernahme von Theorien, die für die Bewirtschaftung materieller Güter mit materiellen Produktionsmitteln entworfen wurden.

Es fällt nicht schwer, die Begriffe der Synergetik[25] auf die Theorie der steigenden Erträge anzuwenden. Die sich durchsetzende Technologie, und der zugehörige Anbieter, bilden einen „Ordner". Eine Vielzahl von Kunden wird „versklavt", diese und darauf aufbauende Technologien einzusetzen.

[25] vgl. Kapitel 3.2 auf Seite 11

#### 5.2.3 Markttheorie, Transaktionskostentheorie und Marktwirtschaft im Unternehmen

Seit den Anfängen der *freien Marktwirtschaft* mit Adam Smith und John Locke hat sich das Konzept bis heute als einzig global gängiges Vorgehen erwiesen. Dies zeigt z. B. der Niedergang der Planwirtschaft in vielen ehemaligen sozialistischen Staaten. Allerdings existiert eine bunte Vielfalt von Umsetzungen der freien Marktwirtschaft. Das Spektrum reicht von sehr liberalen Implementierungen wie in den USA oder Großbritannien bis hin zu stärker geregelten Ansätzen, wie die soziale Marktwirtschaft in Deutschland.

Auf dem in der Marktwirtschaft beschriebenen Markt existieren eine Reihe von Anbietern bestimmter Leistungen. Ihnen gegenüber stehen die Nachfrager nach den Leistungen. Dabei geht die Lehre der *freien Marktwirtschaft* davon aus, dass eine unbegrenzte Nachfrage auf der Seite der Nachfrager existiert und somit Leistungen immer knapp sind. Aus dem Verhältnis von Angebot und Nachfrage ergibt sich der konkrete Preis für die Transaktion von Anbieter zu Nachfrager.

Prinzipiell findet keine Lenkung der Preisfindung von Außen statt. Die Preisfindung erfolgt selbstorganisiert durch die Individuen auf dem Markt. Allerdings muss jedes Individuum abwägen, ob es eine Leistung vom Anbieter beziehen soll oder ob es die Leistung selber erbringt. Eine interne Erstellung einer Leistung erfolgt immer dann, wenn die Kosten für Eigenfertigung niedriger sind als die externen Beschaffungskosten. (vgl. Sta99, S. 420ff) Bei einer genaueren Betrachtung der externen Beschaffung ergibt sich, dass neben den Produktionskosten die Koordinationskosten[26] anfallen. Die Koordinationskosten beinhalten z. B. die Kosten, um sich auf dem freien Markt, aus der Sicht des Nachfragers, über das Angebot zu informieren und den Vertrag mit dem Anbieter abzuschließen. (Sta99, S. 422) Bei einer Eigenfertigung treten neben den reinen Produktionskosten die Transaktionskosten[27] auf. In den letzten Jahren konnten die externen Koordinationskosten beispielsweise durch den Einsatz des Internet (E-Commerce) und Standardsoftware entscheidend gesenkt werden, was zu einer Verlagerung von interner Produktion an externe Anbieter (*Outsourcing*) führte. Allerdings wurde das Outsourcing am Anfang der 90er Jahre stark übertrieben angewendet, so dass heute eine rückläufige Entwicklung zu beobachten ist.

Durch die Eigenfertigung einer Leistung im Unternehmen werden die Marktmechanismen für diese Leistung außer Kraft gesetzt. Im Unternehmen selber findet kein Ausgleich zwischen Angebot und Nachfrage statt. In diesem Zusammenhang drängt sich die Frage auf, warum ganze Staatswirtschaften über Selbstorganisation gelenkt werden können, dies aber für kleinere Organisationen (die Unternehmen) nicht möglich sein soll? Eine mögliche

[26]Der hier vorgestellte Transaktionskostenansatz geht auf Coase um 1937 zurück. Coase erhielt für diese Arbeit 1991 den Nobelpreis.

[27]genauer dazu online unter http://www.olev.de/t/transaktionskost.htm

Antwort kann lauten, dass im Unternehmen zu wenig Individuen vernetzt sind und sich somit keine Selbstorganisation einstellt. Trotzdem existiert eine Reihe von Ansätzen, die versuchen, Marktverhalten im Unternehmen zu etablieren. Ein erster Schritt in dieser Richtung ist das *Profit-Center* (vgl. Sta99, S. 743). Dabei werden einzelne Unternehmensbereiche als autonome eigenverantwortliche Einheiten geführt. Diese Idee kann bis zur *Unternehmensholding* und *strategischen Netzwerken* ausgebaut werden. Durch die Etablierung interner Kunden und der Nachbildung von Netzwerkstrukturen wird unter anderem versucht, Selbstorganisationseffekte zu stimulieren.

## 5.3 Teilbereich Informatik

### 5.3.1 Sozionik

Als erste Anwendung im Teilbereich Informatik wird die *Sozionik* vorgestellt. Die Sozionik begreift sich als „ein neues Forschungsfeld zwischen (Verteilter) Künstlicher Intelligenz und Soziologie, in welchem die Möglichkeiten eines wechselseitigen Konzepttransfers ausgelotet (...) werden“. (Mal01) Die Sozionik wird im Rahmen eines Schwerpunktes[28] der Deutschen Forschungsgemeinschaft untersucht und betrachtet sich dabei selbst als Grundlagenforschung.

Eine mögliche Umsetzung der Sozionik sind die Multi-Agenten-Systeme. Unter Agent versteht man „ein längerfristig arbeitendes Programm, dessen Arbeit als eigenständiges Erledigen von Aufträgen oder Verfolgen von Zielen in Interaktion mit einer Umwelt beschrieben werden kann“. (vgl. Gör00, S. 949) Aus dieser Definition lassen sich einige Eigenschaften ableiten. (nach Gör00; Pit00) Agenten besitzen *Autonomie*, da sie „keiner (unmittelbaren) Steuerung und Kontrolle durch einen Nutzer“ (Gör00, S. 949) unterliegen. Agenten kooperieren und kommunizieren mit anderen Agenten „zur Konstruktion *kollektiver* Problemlösungsstrategien“. (Pit00, S. 19)[29] Diese Eigenschaft wird als *soziales Verhalten* (Gör00, S. 950) bezeichnet. Agenten reagieren auf Veränderungen ihrer Umgebung, bezeichnet als *Reaktivität.* (Gör00, S. 949) Allerdings findet nicht nur eine passive Reaktion auf Umweltänderungen statt, sondern der Agent handelt aus eigenem Antrieb heraus, was als *Proaktivität* (Gör00, S. 950) bezeichnet wird.

Unter einem Multi-Agenten-System versteht man „eine Menge von *interagierenden* Agenten“. (Gör00, S. 996)[30] Durch die Kooperation und Kommunikation der einzelnen Agenten kann man davon ausgehen, dass diese ein kollektives Verhalten zeigen, das nicht auf die individuellen Eigenschaften zurückgeführt werden kann. Die Beobachtung von Emergenz ist somit möglich. Die Idee lässt sich anhand von zellulären Automaten sehr gut nach-

[28] http://www.tu-harburg.de/tbg/Deutsch/SPP/Start_SPP.htm
[29] Hervorhebungen durch Verfasser
[30] Hervorhebungen durch Verfasser

vollziehen.[31] Zelluläre Automaten stellen das dynamische Verhalten diskreter Systeme dar. Zwischen den Einheiten findet keine Kommunikation oder Kooperation statt. Meist besitzen die Einheiten lediglich den Zustand an/aus. Der neue Zustand ergibt sich aus dem aktuellen Zustand und den aktuellen Zuständen anderer Einheiten in der näheren Umgebung. Trotz dieser einfachen Architektur kann hoch komplexes Verhalten wie Schwarmverhalten (ähnlich bei Zugvögeln) beobachtet werden. Eine leicht verständliche Einführung findet man bei Braitenberg (Bra93).

In der Sozionik werden Theorien aus der Soziologie übernommen, die auf Selbstorganisation basieren. Deshalb zeigen die Ergebnisse, z. B. in Form der Multi-Agenten-Systeme, ebenfalls selbstorganisiertes Verhalten.

### 5.3.2 Organic Computing

Einen relativ aktuellen Ansatz stellt das *Organic Computing* dar. Es bezieht sich hauptsächlich auf Hardware. Hauptziel des Organic Computing ist der organische Computer, definiert als „ein selbstorganisierendes System, das sich den jeweiligen Umgebungsbedürfnissen dynamisch anpasst“. (GI03, S. 1) Ein organischer Computer muss „selbst-konfigurierend, selbst-optimierend, selbst-heilend und selbst-schützend“ (GI03, S. 1) sein.

Als Anwendungsbereich wird z. B. das Automobil (Rad04) angeführt. Man geht davon aus, dass die in einem Automobil integrierten Embedded Controller in ihrer Anzahl weiter zunehmen werden und immer wichtigere – teils lebenswichtige – Funktionen, wie Steuerung und Bremsen, übernehmen. Da eine fehlerfreie Funktionsweise nicht garantiert werden kann, wird über den Weg des Organic Computing versucht, eventuell lebensbedrohliche Situationen für den Nutzer zu vermeiden.

Ein weiteres Beispiel (nach GI03, S. 2) ist die Smart Factory. Dabei bilden autonome Roboter spontane Föderationen zur Erledigung anstehender Aufgaben. Beim Ausfall oder Überlastung von Teilsystemen findet eine selbstorganisierte Neuverteilung der Aufgaben statt.

Die Vertreter des Organic Computing weisen darauf hin, dass die nötige Software und Hardware *flexibel* und *adaptiv* sein muss. Wie bei der Sozionik wird nicht versucht die Planungs- und Koordinationsaufgaben zentral zu lösen, sondern eine Vielzahl kleinerer hoch spezialisierter Einheiten sollen im Verbund die anstehenden Aufgaben selbstorganisiert bewältigen.

### 5.3.3 Neuronale Netze

Eine weitere Anwendung, die hier betrachtet wird, sind die künstlichen *neuronalen Netze*. An dieser Stelle wird das Konzept kurz vorgestellt. (vgl. Gör00, S. 73ff) Im Zentrum steht eine Menge von künstlichen Neuronen, die

[31] Unter http://llk.media.mit.edu/projects/emergence/index.html findet sich eine sehr schöne grafische Darstellung.

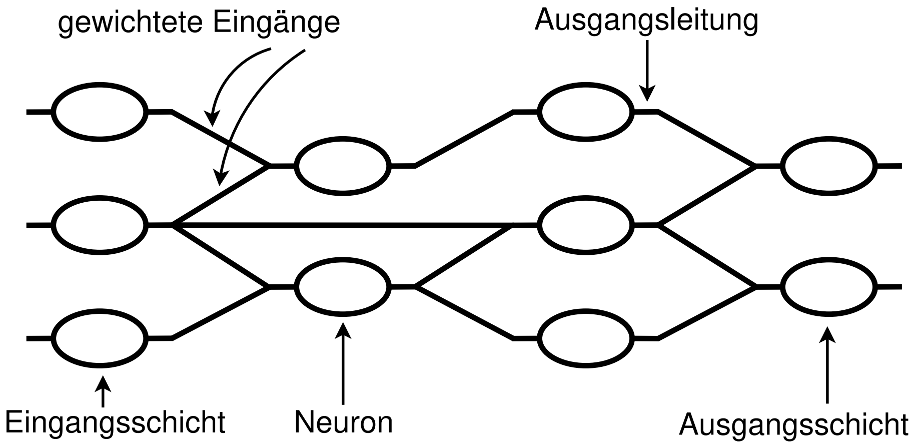


Abbildung 5: Neuronales Netz

untereinander verbunden sind. Jedes Neuron besitzt eine Vielzahl von Eingangsleitungen, die *gewichtet* werden, d. h. jede Eingangsleitung hat einen unterschiedlich starken Einfluss auf das Neuron. Jedes Neuron ermittelt anhand seiner *Aktivierungsfunktion* und den Eingangswerten seinen eigenen Zustand und gibt die entsprechende Information an seine Ausgangsleitung weiter. Diese Ausgangsleitung kann nun wiederum als Eingangsleitung für verschiedene Neuronen dienen. Dadurch entsteht das in Abbildung 5 auf Seite 33 dargestellte Netz.

Einige Neuronen bilden die Eingangsschicht. Über die Eingangsleitungen dieser Neuronen erfolgt die Eingabe. Weiterhin bilden einige Neuronen die Ausgangsschicht. Über die Ausgangsleitungen dieser Neuronen erfolgt die vom neuronalen Netz ermittelte Ausgabe. In der einfachsten Architektur besteht ein neuronales Netz aus lediglich einem Neuron mit einer Vielzahl von Eingangsleitungen. Dieses neuronale Netz kann, da nur eine Ausgangsleitung vorhanden ist, lediglich ja/nein Ergebnisse liefern. Darüber hinaus kann in solch einem neuronalen Netz keine XOR-Verknüpfung abgebildet werden. Deshalb ist man wesentlich später zu mehrschichtigen neuronalen Netzen übergegangen, wie in Abbildung 5 gezeigt. Zwischen Ein- und Ausgabeschicht befindet sich eine Vielzahl von (versteckten) Schichten.

Bevor ein neuronales Netz eine Aufgabe lösen kann, muss es trainiert werden. Durch das Training werden die *Gewichte* an den Eingangsleitungen festgelegt. Bei einigen Varianten von neuronalen Netzen erfolgt weiterhin eine Festlegung des *Schwellenwertes* der Aktivierungsfunktion. Das neuronale Netz lernt anhand von Beispieldaten. Nach der Lernphase kann das neuronale Netz für die eigentliche Aufgabe verwendet werden.

Neuronale Netze sind eine sehr eindrucksvolle Anwendung von Emergenz. Während der Trainingsphase findet eine Selbstorganisation der Netz-

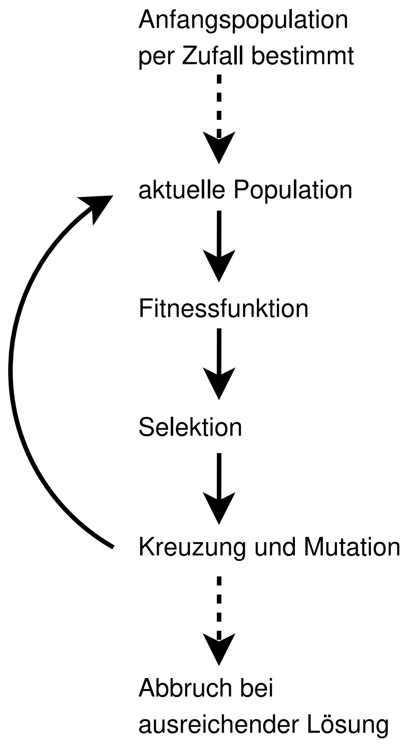


Abbildung 6: Ablauf genetischer Algorithmus

struktur statt. Als Ergebnis können komplexe Aufgaben wie Muster- und Spracherkennung gelöst werden. Ein interessanter Beitrag in diesem Zusammenhang ist der *synergetische Computer* von Haken. (z. B. bei HW90) Er formuliert damit ein sehr ähnliches Vorgehen, allerdings geht Haken bei der Betrachtung von der Makroebene (Gesamtverhalten) und nicht vom einzelnen Individuum (Neuronen) aus.

### 5.3.4 Genetische Algorithmen

Abschließend werden die *genetischen Algorithmen* vorgestellt. (z. B. nach GS02, S. 243ff) Den genetischen Algorithmen liegt die evolutionäre Entwicklungstheorie nach Darwin als Modellvorstellung zugrunde. Es wird dabei versucht, dass sich eine Lösung für ein Problem mit der Zeit entwickelt und verbessert. Dazu werden am Anfang zufällig eine Reihe von Lösungen, die *Individuen*, ermittelt, die das Problem bereits lösen (siehe dazu Abbildung 6 auf Seite 34). Es erfolgt eine Bewertung der einzelnen Individuen mithilfe der *Fitnessfunktion*. Aus den besten Individuen werden *Nachfolger* erzeugt, indem die Individuen Teile ihrer *Gene* an die *Kinder vererben*. Dabei kommt z. B. die *Kreuzung* zum Einsatz. Um den Genpool der Individuen aufzufrischen, werden spontane *Mutationen* zugelassen. Die entstandenen

Kinder werden wiederum bewertet. Damit beginnt der Prozess erneut und wird solange iterativ fortgesetzt, bis sich eine akzeptable Lösung ergibt.

Von besonderem Interesse im Hinblick auf diese Arbeit ist, dass bewusst der Zufall in Form von *Mutationen* genutzt wird, um eine Lösung zu erhalten. Weiterhin liegt bei den genetischen Algorithmen ein nicht-deterministisches Vorgehen vor. Es ist prinzipiell vorher nicht genau bestimmbar, wie viel *Generationen* von Individuen benötigt werden, um eine akzeptable Lösung hervorzubringen und wie diese Lösung am Ende aussieht. Das Hauptproblem bei der Implementierung von genetischen Algorithmen besteht in einer effizienten Kodierung der Lösungen als Individuen.[32]

Neben der Anwendung von genetischen Algorithmen bei Such- und Optimierungsproblemen werden diese zur Bestimmung der Gewichte von neuronalen Netzen benutzt. An diesem Vorgehen kann man erkennen, dass in der Praxis oftmals mehrere Verfahren kombiniert werden, die auf Selbstorganisation, Nicht-Linearität und Nicht-Determinismus basieren.

## 5.4 Zuordnung zur Systematik

Wie bereits in Kapitel 4.4 auf Seite 24 angedeutet, wird nun versucht, die in diesem Kapitel vorgestellten Verfahren und Theorien zu systematisieren. Dabei erfolgt immer eine Zuordnung zu genau einer Kategorie, selbst wenn dies teilweise nicht immer eindeutig möglich ist.

**Bewältigung der Anforderungen einer emergenten Welt:** Die *lernende Organisation* sowie das *Wissensmanagement* dienen der Bewältigung einer sich ständig ändernden Welt. Es erfolgt somit eine Anpassung an die Welt, aber keine bewusste Gestaltung. Die *Theorie steigender Erträge* liefert ein Erklärungsmodell für das Marktgeschehen einer emergenten Wirtschaftswelt. Aus dieser Theorie können konkrete Handlungsanweisungen abgeleitet werden, die eine Ausnutzung bewirken. In diesem Fall müsste die Theorie in die zweite Kategorie, die Gestaltung einer emergenten Welt, eingeordnet werden. Das *virtuelles Unternehmen* ist ebenfalls ein Grenzgänger. Es ermöglicht gerade dem Mittelstand in einem sehr dynamischen Umfeld komplexe Leistungen zu erbringen und damit in der Wirtschaftswelt zu bestehen. Andererseits ist das virtuelle Unternehmen ein Gestaltungsmittel, mit dem Emergenz hervorgerufen werden kann.

**Gestaltung einer emergenten Welt:** Das *fraktale Unternehmen*, also die Nutzung von Marktmechanismen und kleinen autonomen Einheiten im Unternehmen, ist der Versuch, emergente Systeme zu schaffen. Es findet eine Vernetzung verschiedener relativ autonomer Individuen statt, zur

[32] D. h., wie kann man die Lösung eines Problems als eine Kette von einzelnen Bits so kodieren, dass Selektion und Mutation sinnvoll angewendet werden können.

Erbringung einer gemeinsamen Gesamtleistung. Die *freie Marktwirtschaft* setzt dieses Prinzip auf der Makroebene um. Die konkretesten Gestaltungen emergenter Systeme sind die *Sozionik*, das *Organic Computing*, die *neuronalen Netze* und die *genetischen Algorithmen*. Hier findet sich eine vollständige Abkehr vom mechanischen Weltbild. Es wird versucht, Probleme durch Hervorrufung von Nicht-Linearität und Nicht-Determinismus zu lösen.

# 6 Klassische Softwareentwicklung

## 6.1 Einleitung

Nach der Analyse von Emergenz und der Darstellung von Emergenzanwendungen in den vorherigen Abschnitten dieser Arbeit, wird in den folgenden Kapiteln eine eventuell mögliche Anwendung von Emergenz im Bereich Softwareentwicklung untersucht. Dazu ist es sinnvoll, zunächst die Softwareentwicklung allgemein zu charakterisieren. Darauf aufbauend wird dargestellt, welche Grundannahmen und Lösungsvorschläge bisher für die Softwareentwicklung angewendet werden. Diese Darstellung erfolgt in diesem Kapitel kritiklos. Im folgenden Kapitel wird dann die agile Softwareentwicklung näher untersucht. Dabei ist von Interesse, ob die agilen Methoden für eine Umsetzung von Emergenz in der Softwareentwicklung sorgen können.

## 6.2 Softwareentwicklung allgemein

Software ist das Produkt des Softwareherstellers. (für diesen Abschnitt vgl. z. B. ZBGK01, S. 17ff) An den Hersteller tritt ein Kunde heran bzw. der Hersteller spricht einen potenziellen Kunden an. Dabei spielt es im Rahmen dieser Arbeit keine Rolle, ob es sich um einen externen oder internen Kunden handelt. Der Kunde hat ein konkretes Problem, dessen Lösung der Hersteller als Problemlösung zu erarbeiten hat. Die Problemlösung kann neben dem eigentlichen Softwareprodukt weiterhin Hardware und Dienstleistungen wie Beratung und Schulung umfassen. Der Hersteller muss das Problem des Kunden analysieren und eine mögliche Lösung beschreiben. Da es bis heute nicht möglich ist Software automatisiert in Serienfertigung herzustellen, muss die Softwareentwicklung als Einzelfertigung betrachtet werden. Selbst bei der Entwicklung von Standardsoftware wie Bürosoftware, Finanzsoftware und Betriebssystemen ist die erstmalige Entwicklung dieser Software eine Einzelfertigung gegenüber einem internen Kunden. Für die Bewältigung der Einzelfertigung hat sich das Vorgehen im Rahmen eines Projektes als Managementform in vielen verschiedenen Disziplinen bewährt. Deshalb wird Softwareentwicklung ebenfalls als Projekt durchgeführt. Ein Projekt ist dabei gekennzeichnet durch folgende Eigenschaften (nach ZBGK01, S. 27f):

1. einmaliges Vorhaben
2. zeitlich begrenzt
3. klare Ziele zu Beginn
4. neuartige und unbekannte Probleme werden gelöst
5. unterschiedliche Methoden bei verschiedenen Projekten
6. Zusammenarbeit von Personen aus unterschiedlichen Fachgebieten

7. besonderes Risiko (z. B. finanzielle Folgen beim Scheitern des Projektes für das Unternehmen)

8. eigenes Budget

Der in Eigenschaft 6 angesprochene Zusammenschluss wird als Projektteam bezeichnet. Jedem Mitarbeiter des Projektteams werden verschiedene Rollen zugeordnet, wie z. B. die Rolle des Projektleiters. Der Projektleiter vertritt das Projekt nach Außen und stellt somit die Schnittstelle zwischen Projekt und Kunde aber auch zwischen Projekt und Management des Herstellers dar. Damit der Projektleiter seiner Verantwortung gegenüber Kunde und Management gerecht werden kann, erhält er im Projektteam eine Sonderstellung, indem ihm Entscheidungsbefugnis zugeordnet wird. Je nach Größe des Projektteams werden unterschiedliche Organisationsmodelle angewendet.

Kunde und Management des Herstellers vereinbaren bei Vertragsabschluss meist den Produktpreis, den Auslieferungstermin, den Funktionsumfang sowie die Qualitätsanforderungen des Produktes. Damit für den Hersteller das Projekt ein wirtschaftlicher Erfolg wird, muss der Projektleiter versuchen das Projekt unter Erfüllung von Auslieferungstermin, Funktionsumfang und Qualität zu niedrigeren Kosten, als mit dem Produktpreis vereinbart, abzuschließen.

Um das Projekt erfolgreich durchführen zu können, wird ein Softwareentwicklungsprozess angewendet. Darunter versteht man „die Summe aus einem Vorgehensmodell, den Tätigkeiten und Aktivitäten sowie den angewandten Methoden." (ZBGK01, S. 25) Das Vorgehensmodell „ist eine Beschreibung einer koordinierten Vorgehensweise bei der Abwicklung eines Vorhabens." (Ver02, S. 29) Dabei legt das Vorgehensmodell eine Reihe von Aktivitäten fest sowie deren Input und Output (Artefakte). Weiterhin erfolgt eine „feste Zuordnung von Rollen (. . . ), die die jeweilige Aktivität ausüben." (Ver02, S. 29) Die Rollen, und somit die Verantwortung für die Aktivitäten, werden den einzelnen Mitgliedern des Projektteams zugeordnet.

Es existiert eine Vielzahl von Vorgehensmodellen. Im nächsten Abschnitt werden nun die „klassischen" Vorgehensmodelle und die dahinter liegende Metapher vorgestellt.

## 6.3 Vorgehensmodelle der Klassischen Softwareentwicklung

In der Softwareentwicklung sind eine Reihe von Vorgehensmodellen zur Anwendung gekommen. Diese entstanden oft aus der Erfahrung von Projekten und wurden dann allgemeingültig formuliert. Vereinfachend lassen sich drei Klassen von Vorgehensmodellen erkennen:

- „Code and Fix"

- Wasserfallmodell und V-Modell

- Iterativ inkrementelles und plangetriebenes (plan driven) Vorgehen

Die Aufzählung erfolgte in chronologischer Reihenfolge der Entstehung. Es werden nun kurz die einzelnen Klassen beschrieben.

### 6.3.1 Vorgehensmodell „Code and Fix“

„Code and Fix“ steht für „Programmierung und Fehlerbehebung“, teilweise auch bekannt als „Build-and-Fix“ Zyklus. Darunter wird ein völlig unstrukturiertes Vorgehen verstanden. Entstanden ist dieses Vorgehensmodell in der Anfangszeit der Rechentechnik, als Software meist von einem einzigen Entwickler produziert wurde.

Der Entwickler beginnt direkt mit der Implementierung des Systems. Die Entwicklung wird solange fortgesetzt, bis das System den Vorstellungen des Programmierers entspricht. Fehler werden beim Auftreten durch den Programmierer behoben. Dieser Ansatz ist unstrukturiert, es wird kaum Dokumentation erstellt und Analyse und Entwurf werden ebenfalls fast komplett ausgespart. (vgl. ZBGK01, S. 45) Das Vorgehensmodell „Code and Fix“ ist dann erfolgsversprechend, wenn Entwickler und späterer Nutzer Rollen einer einzigen Person sind. Auch wenn dieses Vorgehensmodell durchaus qualitativ hochwertige Software hervorbringen kann, ist das Vorgehen in der professionellen Softwareentwicklung abzulehnen, da eine Wartung oder Weiterentwicklung der Software meist nur durch den ursprünglichen Entwickler möglich ist.

### 6.3.2 Vorgehensmodell Wasserfallmodell und V-Modell

Das Wasserfallmodell, und die in Deutschland Anfang der 90er Jahre erfolgte Weiterentwicklung hin zum V-Modell, versuchen den Software-Lebenszyklus umzusetzen. Der Software-Lebenszyklus wird definiert als Abfolge folgender Phasen:

1. Anforderungen
2. Analyse
3. Entwurf (Design)
4. Implementierung
5. Test
6. Inbetriebnahme
7. Wartung

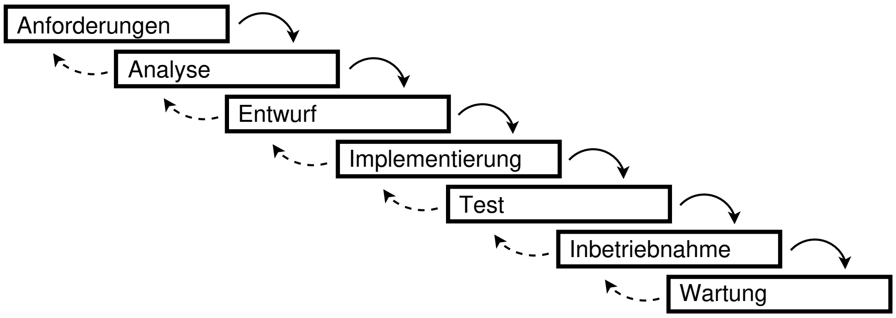


Abbildung 7: Wasserfallmodell

Treten neue oder geänderte Anforderungen an die Software auf, müssen diese in einem komplett neuen Zyklus umgesetzt werden. Dies ist ein sehr starres Vorgehen, welches wenig Flexibilität bietet. Kritisch wird dieses Vorgehen, wenn in einer Phase erkannt wird, dass die vorherige Phase unvollständige Ergebnisse geliefert hat. (vgl. ZBGK01, S. 45f)

Im Wasserfallmodell ist deshalb eine Rückkoppelung zur vorherigen Phase vorgesehen. Damit ergibt sich das klassische Bild wie in Abbildung 7 auf Seite 40 gezeigt. Das V-Modell wurde aus dem Wasserfallmodell abgeleitet. Prinzipiell ist es sowohl zur Bearbeitung von Softwareprojekten aber auch für die Bearbeitung anderer Projekte ausgelegt. Das V-Modell entstand im Bereich der Rüstungsindustrie und die Anwendung des V-Modell war zeitweise Voraussetzung bei der Vergabe von Rüstungsaufträgen. Allerdings wird das V-Modell heute nicht mehr weiterentwickelt.[33]

Das V-Modell, an dieser Stelle wird nur das Submodell Systementwicklung betrachtet, gliedert die Entwicklung in folgende sechs Phasen:

1. Analyse (der Anforderungen)
2. Systementwurf
3. Feinentwurf und Implementierung
4. Modultest
5. Systemintegration
6. Systemabnahme

Die Phasen 1 bis 3 stehen auf der linken Seite des imaginären Buchstabens „V“, die Phasen 4 bis 6 auf der rechten Seite. Es ergibt sich das in Abbildung 8 auf Seite 41 gezeigte Bild. Phase 4 (Modultest) überprüft das Artefakt der Phase 3 (Feinentwurf und Implementierung), Phase 5 (Systemintegration)

[33] Die wohl aktuellste Variante steht auf der Seite http://www.cocoo.de/ zur Verfügung.

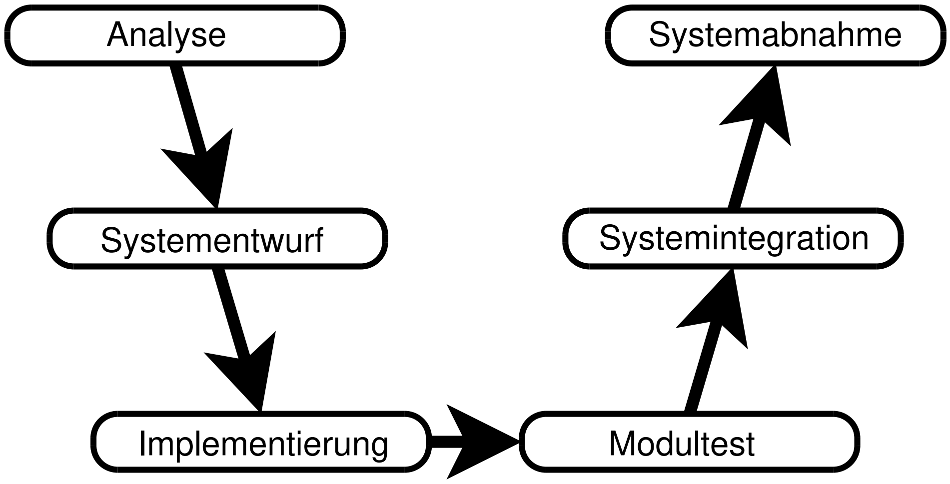


Abbildung 8: Schichten des V-Modell

überprüft das Artefakt der Phase 2 (Systementwurf) und Phase 6 (Systemabnahme) überprüft das Artefakt der Phase 1 (Anforderungsanalyse). Sollte sich bei diesen Überprüfungen eine Unstimmigkeit ergeben, muss zur überprüften Phase zurückgekehrt werden. Bei einem gescheiterten Modultest ist der Rückschritt zu Phase 3 kein größeres Problem. Wird allerdings in Phase 6 bei der Abnahmeprüfung festgestellt, dass die Anforderungen in Phase 1 unzureichend verstanden wurden, dürfte dies in den meisten Fällen zum Projektabbruch und damit zum Scheitern des Projektes führen. Es ist dabei zu beachten, dass in großen Projekten zwischen Analyse (Phase 1) und Systemabnahme (Phase 6) durchaus mehrere Jahre liegen können.

Wasserfallmodell und V-Modell stellen einen entscheidenden Vorteil gegenüber dem „Code and Fix“ Vorgehen dar. Die klare Strukturierung in Phasen sowie die Festlegung der von den Phasen zu erzeugenden Outputs (Artefakte) sorgt z. B. für eine Dokumentation des Vorgehens. Weiterhin ist eine Koordinierung mehrerer Entwickler und Projektanten möglich. Als Nachteil ist die mangelnde Flexibilität anzusehen. Auf ändernde Anforderungen kann nur schlecht reagiert werden, was hohe Kosten verursacht. (vlg. ZBGK01, S. 47)

### 6.3.3 Iterativ inkrementelle und plangetriebene Vorgehensmodelle

Um den Hauptkritikpunkt einer unzureichenden Flexibilität von Wasserfallmodell und V-Modell zu umgehen, wurde die iterativ inkrementelle Softwareentwicklung entwickelt. Diese wird häufig auch als evolutionäre Softwareentwicklung bezeichnet. Das Spiralmodell (vgl. z. B. ZBGK01, S. 47ff) gilt als Vorläufer, da es zwar einen iterativen Ansatz umsetzt, aber nicht inkrementell vorgeht. Im Spiralmodell werden die folgenden vier Phasen bis zum Projektabschluss wiederholt:

1. *Zielbestimmung* für diese Iteration
2. *Risikoanalyse*, mögliche Alternativen finden und bewerten
3. *Ausführung* der besten Alternative
4. *Planung der nächsten Iteration*, Begutachtung (Review) der Ergebnisse dieser Iteration

Die Liste der Phasen verdeutlicht, dass es sich um ein wesentlich abstrakteres Vorgehensmodell handelt. Speziell die Phase 3 jeder Iteration kann stark unterschiedlich ausgeprägt sein. Zur Durchführung der Phase 3 kann das V-Modell oder Wasserfallmodell verwendet werden.

Bei einer inkrementellen Softwareentwicklung erfolgt die Entwicklung der Software schrittweise. Der komplette Entwicklungszyklus wird dazu mehrfach wiederholt (iterativ). Somit wird das Softwareprodukt während der Entwicklung schrittweise umfangreicher. Der Gedanke der evolutionären bzw. iterativ inkrementellen Softwareentwicklung wurde in einer Vielzahl von weiteren Vorgehensmodellen umgesetzt. In diesem Zusammenhang sei z. B. der Rational Unified Process erwähnt. Heutzutage aktuell angewendete Vorgehensmodelle, wie der Rational Unified Process, werden als plangetrieben (plan-driven) bezeichnet. Dies ist auf die umfangreichen Regelwerke dieser Vorgehensmodelle zurückzuführen. Dabei wird vom Anwender des Vorgehensmodells gefordert, dieses an die unternehmensspezifische Situation und Prozesse anzupassen.[34] Zur Umsetzung dieser Vorgehensmodelle stehen umfangreiche Werkzeugsammlungen und Beratungsleistungen zur Verfügung. Aber auch die Anwendung dieser Vorgehensmodelle ist keine Garantie für erfolgreiche Softwareprojekte.

Bevor im nächsten Kapitel die Kritik an diesen Vorgehensmodellen aus der Sicht der agilen Softwareentwicklung genannt wird, erfolgt nun die Herausarbeitung der den hier vorgestellten Vorgehensmodellen zugrunde liegende Basismetapher.

## 6.4 Softwareentwicklung als Ingenieurdisziplin

Prinzipiell wird Software nach den klassischen Vorgehensmodellen in folgenden sechs Phasen im Rahmen eines Projekts entwickelt:

1. Analyse
2. Entwurf (Design)
3. Implementierung

[34] Innerhalb des Rational Unified Process wird dies als Tailoring bezeichnet.

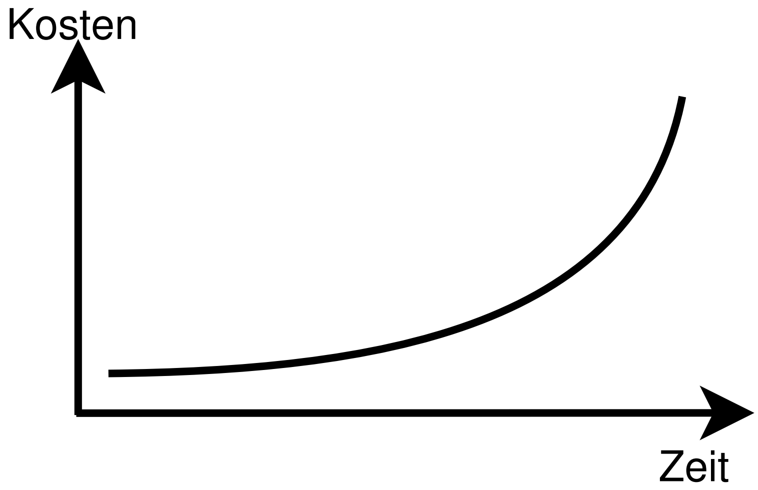


Abbildung 9: zeitliche Abhängigkeit der Kosten für Änderungen nach Boehm

4. Test
5. Inbetriebnahme
6. Wartung

Während im Wasserfallmodell und V-Modell diese Phasen mit kurzen Rückkoppelungen im Idealfall nur einmal durchlaufen werden, erfolgt eine iterative Wiederholung der Phasen in der evolutionären Softwareentwicklung. Entscheidend dabei ist, dass während der Analyse möglichst alle Anforderungen an die zu entwickelnde Software erkannt werden. Später auftretende Änderungen können nach dem Entwurf der Software oft nur schlecht oder unter hohem Kostenaufwand berücksichtigt werden. Aus dieser Darstellung ergibt sich die auf Barry Boehm (vgl. z. B. Col03) zurückgehende Kostenkurve, wie in Abbildung 9 auf Seite 43 zu sehen. Diese zeigt, dass später auftretende Anforderungen wesentlich höhere Kosten (exponentieller Anstieg) verursachen, als frühzeitig erkannte Anforderungen. Deshalb wird in der klassischen Softwareentwicklung der Schwerpunkt auf die Analyse gelegt, um möglichst alle Anforderungen zu erkennen. Darin inbegriffen ist die These, dass die Softwarearchitektur nur schwer änderbar ist und somit nur unter hohen Kosten Änderungen der Anforderungen berücksichtigt werden können.

Dieses Vorgehen findet sich in einer Reihe von Ingenieurdisziplinen wieder. Für die Softwareentwicklung wird oftmals die Metapher des Hausbaus benutzt. Bevor mit dem eigentlichen Bau begonnen werden kann, muss zuerst eine Architektur (Entwurf) anhand der Anforderungen der späteren Bewohner erarbeitet werden. Diese Architektur kann dann durch eine Baufirma realisiert werden. Sollten sich während der Bauausführung Änderungen ergeben, können diese nur schwer berücksichtigt werden, da größere Änderungen eine Überarbeitung der Architektur erfordern würden.

Die Baumetapher hat sich als sehr wertvoll für die Softwareentwicklung erwiesen. Kein Architekt bzw. Konstrukteur entwickelt heute jedes Haus oder jede Brücke komplett neu. Vielmehr wird auf einen reichen Erfahrungsschatz zurückgegriffen und bekannte Elemente werden kombiniert. In diesem Zusammenhang entstand der Gedanke wieder verwendbarer Softwarekomponenten. So können heute enorme Produktivitätssteigerungen durch den Einsatz von Frameworks wie Microsoft .NET, Suns Java Klassen oder der Qt Bibliothek von Trolltech erreicht werden. Weiterhin wurde ein Katalog mit möglichen Software Architekturen und Entwurfsmustern erstellt. (vgl. Gam96) Als Vorbild für die Entwurfsmuster werden die Architekturmuster des Architekturprofessors Christopher Alexander zitiert.

Jede Phase in der klassischen Softwareentwicklung erzeugt definierte Artefakte. Diese Artefakte dokumentieren den kompletten Prozess und das Produkt. Anhand dieser Dokumentation kann das Projekt nachvollzogen und kontrolliert werden. Eine Einarbeitung eines neuen Mitarbeiters ist mit dieser Dokumentation möglich. Darüber hinaus dienen die bereits produzierten Artefakte als Arbeitsgrundlage der folgenden Phasen. Im Idealfall benötigt ein Mitarbeiter kein Vorwissen aus den vergangenen Phasen um seine Arbeit fortzusetzen. In der Endkonsequenz werden Mitarbeiter dadurch theoretisch leichter austauschbar. Dies führt zu einer Geringschätzung des Mitarbeiters und es wird somit eher in Werkzeuge und Prozesse als in Mitarbeiter und deren Qualifikation investiert.

Aufgrund der klar definierten Ergebnisse einer Phase kann objektiv entschieden werden, ob eine Phase in der Softwareentwicklung erfolgreich beendet ist. Ein umfangreiches Controlling ist möglich. Es wurden eine Vielzahl von Softwaremetriken zur Beurteilung der Qualität der produzierten Software, aber auch zur Beurteilung der Qualität des Projektes insgesamt, entwickelt. Diese Metriken unterstützen zukünftige Projekte z. B. bei Aufwandsschätzungen. Insgesamt gesehen findet eine ständige Optimierung der Softwareentwicklung statt. Diese Optimierung ist notwendig, um die stetig komplexer werdenden Anforderungen bewältigen zu können.

Weiterhin existiert die Tendenz eine möglichst hohe Automatisierung, z. B. im Rahmen der Model Driven Architectur[35], während der Software-

[35]Im Rahmen der Model Driven Architectur wird die Geschäftslogik in einem plattformunabhängigen Modell (PIM) beschrieben. Dieses Modell kann in ein Modell für eine konkrete Plattform (PSM) transformiert werden. Für diese Transformation müssen teils weitere Informationen dem Modell hinzugefügt werden, ansonsten erfolgt die Transformation weitestgehend automatisiert. Diese Informationen können wiederum verwendet werden, falls das plattformunabhängige Modell für eine weitere Plattform transformiert werden soll. Die Modellierung der Geschäftslogik wird somit komplett von der Modellierung der tatsächlichen Anwendung getrennt. Für die Modellierung wird z. B. der Einsatz der Unified Modelling Language (UML) und entsprechender Modellierungswerkzeuge vor-

entwicklung zu erreichen. Neben einer Kostenreduktion durch Einsparung von Mitarbeitern wird eine Qualitätssteigerung angestrebt. Dazu sind hohe Investitionen in Werkzeuge notwendig, vergleichbar dem Aufbau von vollständig automatisierten Fertigungsstraßen in der Güterindustrie. Auch hier sind die Parallelen zu anderen Ingenieurdisziplinen unverkennbar.

All diesen verschiedenen Ausprägungen der klassischen Softwareentwicklung liegen letztendlich zwei Grundannahmen zugrunde, die mit dem Begriff *mechanisches Weltbild*[36] nach Newton zusammengefasst werden können. Die zwei Grundannahmen lauten:

1. Determinismus
2. Linearität

Es wird jetzt gezeigt, worin sich diese Grundannahmen konkret manifestieren.

### 6.4.1 Determinismus in der klassischen Softwareentwicklung

Wendet man den Begriff Determinismus[37] auf die Softwareentwicklung an, dann geht man von einem prinzipiell genau vorherbestimmbaren Projektverlauf aus. Man spricht in diesem Zusammenhang von konzeptioneller Sicherheit. Nur unter dieser Annahme ist eine Analyse sinnvoll durchführbar. Denn können die Anforderungen genau vorherbestimmt werden, dann kann das Risiko von später auftretenden Anforderungsänderungen eliminiert werden. Somit ist eine Konzentration auf die stetige Verbesserung der Analyse sinnvoll.

Der Determinismus zeigt sich weiterhin im Projektmanagement der Softwareentwicklung. Es werden eine Reihe von Terminen festgelegt, zu denen ein bestimmter Funktionsumfang für das Produkt erwartet wird. Diese Punkte im Projektplan werden als Meilensteine bezeichnet. Eine solch genaue Planung ist nur möglich, wenn man davon ausgeht, dass sich die sorgfältig erarbeiteten Pläne in die Realität umsetzen lassen und man somit die Zukunft durch Planung mit einer abschätzbaren Erfolgswahrscheinlichkeit gestalten kann.

### 6.4.2 Linearität in der klassischen Softwareentwicklung

Wendet man den Begriff Linearität[38] auf die Softwareentwicklung an, dann führen kleine Abweichungen vom Plan zu überschaubaren Konsequenzen für

geschlagen.

[36]vgl. Kapitel 4.2 auf Seite 21

[37]inhaltlich als Umkehrung der Definition in Kapitel 3.5 auf Seite 19

[38]inhaltlich als Umkehrung der Definition in Kapitel 3.5 auf Seite 19

das Gesamtprojekt. Linearität bildet ebenfalls die Grundlage für die Programmierung und Fehlerbehebung. Danach haben kleine Änderungen ebenfalls keine großen Auswirkungen. Weiterhin liegt die Annahme von Linearität einer späten Integration zugrunde. In der Softwareentwicklung wird die Integration der Teilsysteme erst meist kurz vor der Auslieferung bzw. Inbetriebnahme durchgeführt. Nichtlineare Rückkoppelungseffekte werden dabei meist nicht betrachtet.

Wenn Linearität für die Softwareentwicklung gilt, dann entwickeln sich ähnliche Projekte ähnlich. Man kann somit auf den Erfahrungen aus anderen Projekten aufbauen und diese Erfahrungen als Vorlage für neue Projekte verwenden.

## 6.5 Zusammenfassung klassische Softwareentwicklung

Es konnte gezeigt werden, dass der klassischen Softwareentwicklung das mechanische Weltbild zugrunde liegt. Eine Anwendung der Begriffe *Linearität* und *Determinismus* ist zwingend. Linearität und Determinismus verhindern allerdings das Auftreten von Emergenz und führen somit nicht zu Selbstorganisation. Im nun folgendem Kapitel wird untersucht, ob die agile Softwareentwicklung Emergenz ermöglichen kann.

# 7 Agile Softwareentwicklung

## 7.1 Einleitung

In diesem Kapitel wird die agile Softwareentwicklung diskutiert. Dazu werden am Anfang zwei Vertreter (*Extreme Programming* und *Methodikfamilie Crystal*) näher vorgestellt. Anschließend wird das *Manifest der agilen Softwareentwicklung* vorgestellt und der Begriff *Agilität* wird näher untersucht. Dabei wird das zugrunde liegende Weltbild identifiziert. Abschließend wird *Emergent Design* näher betrachtet und die vorgestellten agilen Verfahren werden in die Systematik eingeordnet.

## 7.2 Agiler Vertreter: Extreme Programming

### 7.2.1 Einleitung

Extreme Programming[39] ist der bekannteste agile Softwareentwicklungsprozess. Auf den ersten Blick rückt Extreme Programming die eigentliche Implementierung in den Mittelpunkt. Dadurch erscheint Extreme Programming für viele Programmierer sehr interessant, da er den Programmierer von den oft als lästig empfundenen Formalien wie Dokumentationserstellung befreit. Extreme Programming wirkt sehr radikal, da es mit einer Reihe von bisherigen Grundannahmen bricht. Dabei gibt Extreme Programming konkrete Handlungsanweisung zur Gestaltung der einzelnen Arbeitsschritte vor. Die Autoren[40] betonen, dass nur eine vollständige Umsetzung aller von Extreme Programming vorgegebenen Prinzipien zu einer erfolgreichen Anwendung von Extreme Programming führen wird. (vgl. Bec00, S. 63) Extreme Programming ist es durch seine Popularität gelungen, die Aufmerksamkeit des Fachpublikums auf die agilen Methodiken zu lenken. Im Rahmen der agilen Softwareentwicklung wird in der Literatur von *Methodiken* anstatt Vorgehensmodellen gesprochen. Eine Methodik ist danach eine begriffliche Zusammenfassung für Prozesse und Methoden. In dieser Arbeit wird Methodik als Synonym für *Vorgehensmodell* verwendet. Im nun folgenden Abschnitt wird Extreme Programming umfassend dargestellt.

### 7.2.2 Grundwerte

Extreme Programming formuliert vier Grundwerte (vgl. Bec00, S. 29ff), aus denen die 12 Grundpraktiken abgeleitet werden. Die vier Grundwerte dienen als Basis für eine Unternehmenskultur, in der Extreme Programming erfolgreich umgesetzt werden kann. Die vier Grundwerte lauten:

- Kommunikation

[39] Extreme Programming verfügt über eine große Gemeinde von Anhängern. Zahlreiche Internetseiten stellen die Thematik dar, wie z. B. http://www.extremeprogramming.org/.

[40] hauptsächlich Kent Beck, Ward Cunningham und Ron Jeffries

- Einfachheit
- Rückmeldung (Feedback)
- Mut

Extreme Programming versucht durch eine Vielzahl von Praktiken die *Kommunikation* zwischen Entwickler, Kunde und Management zu sichern. Kommunikation wird als essentiell angesehen, damit überhaupt eine Zusammenarbeit gelingen kann. Dabei muss ständig versucht werden, die Kommunikation aufrecht zu halten, denn Kommunikation ist störanfällig. Damit die Kommunikation gewährleistet werden kann, schlägt Extreme Programming den Einsatz eines Coaches vor, der bei Kommunikationsstörungen aktiv eingreift.

Extreme Programming fordert *Einfachheit*. Es soll stets die einfachste Lösung für ein Problem angestrebt werden. Ein Vorausdenken über eventuell zukünftig notwendige Programmdetails während der Programmierung ist nicht erwünscht, da Extreme Programming davon ausgeht, dass diese Vorwegnahme in vielen Fällen falsch ist und damit unnötige Kosten verursacht. Durch ständige Kommunikation erkennen die Entwickler die einfachste Lösung für ein Problem. Weiterhin soll die Architektur möglichst einfach gestaltet werden, denn „je einfacher ein System ist, desto weniger muss darüber mitgeteilt werden". (Bec00, S. 31) Extreme Programming selbst lehnt komplexe Vorgehensmodelle wie den Rational Unified Process ab und fordert stattdessen ein einfaches Regelwerk. Diese Forderung wird auf Extreme Programming selbst angewandt, weshalb die Darstellung der Prinzipien und Grundwerte von Extreme Programming auf wenigen Seiten möglich ist.

Weiterhin fordert Extreme Programming *Feedback*. Feedback z. B. durch Tests soll dem Programmierer über den aktuellen Systemzustand informieren. So können Defekte sofort erkannt und behoben werden. Weiterhin soll der Kunde ebenfalls über den aktuellen Stand des Projektes informiert sein, damit er eingreifen kann. Auf der anderen Seite erhält der Kunde Feedback über seine Anforderungen, inwieweit diese für die Entwickler verständlich sind und mit welchem Aufwand für die Umsetzung zu rechnen ist. Dem Kunden wird frühzeitig die Umsetzung seiner Anforderungen als laufendes System vorgeführt, damit er unmittelbar Feedback zur implementierten Lösung geben kann.

*Mut* bedarf es, um diese Grundwerte konsequent umzusetzen. Man wird bei der Umsetzung mit einer Reihe von Regeln der eigenen Organisation brechen müssen. So fordert Extreme Programming z. B. nicht funktionierenden Code[41] zu verwerfen. Weiterhin muss zum Kunden eine ehrliche und

[41] Unter dem Begriff Code wird in dieser Arbeit der Quelltext in einer Programmiersprache sowie Datenbankskripte, Schnittstellenbeschreibungen für Web Services usw. verstanden.

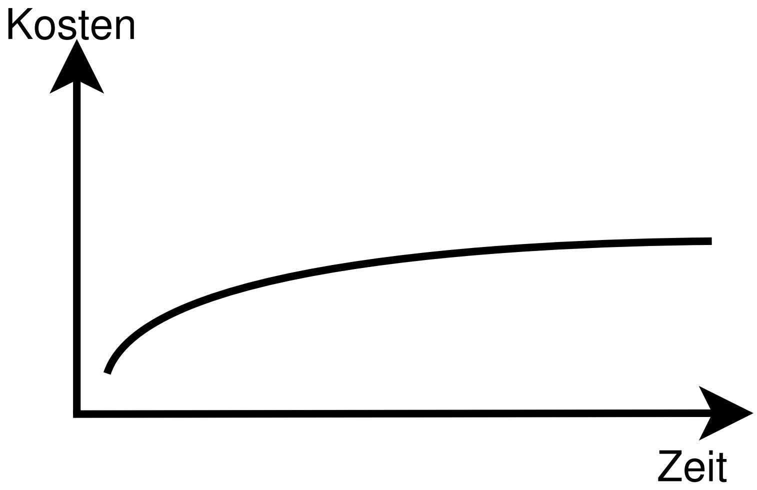


Abbildung 10: zeitliche Abhängigkeit der Kosten für Änderungen nach Beck

vertrauensvolle Beziehung aufgebaut werden. Der Kunde ist davon zu unterrichten, wenn während der Entwicklung z. B. Verzögerungen auftreten. Dieses Vorgehen erfordert Mut.

Diese vier Grundwerte arbeiten zusammen und unterstützen sich gegenseitig. Mit Sicherheit wäre es möglich, die Grundwerte mit anderen Begriffen zu formulieren. Neben den Grundwerten existiert die These, dass späte Änderungen der Anforderungen nicht wesentlich höhere Kosten verursachen, als frühzeitig bekannte Änderungen. Diese These wird im folgenden Abschnitt dargestellt.

### 7.2.3 Umkehrung der Kostenkurve

Extreme Programming stellt die These auf, dass die Kostenkurve[42] nach Barry Boehm nicht (mehr) gültig ist. Durch den Einsatz von modernen Technologien und Programmierverfahren kann die Kostenkurve quasi umgekehrt werden. (vgl. Bec00, S. 21ff) Es ergibt sich der in Abbildung 10 auf Seite 49 gezeigte Graph.

Die Schlussfolgerung aus dieser These lautet, dass eine ständige Änderung der Software jederzeit möglich ist und prinzipiell Kosten auf ähnlichem Niveau verursacht. Deshalb müssen nicht alle Anforderungen vor Beginn des Entwurfs und der Implementierung feststehen. Es können jederzeit neue Anforderungen durch den Kunden aufgestellt werden, alte Anforderungen geändert werden oder noch nicht implementierte Anforderungen gestrichen werden.

[42] siehe Kapitel 6.4 auf Seite 43

Da jederzeit Änderungen an den Anforderungen berücksichtigt werden können, fordert Extreme Programming den Kunden auf, Anforderungen immer dann zu ändern, wenn die Änderung einen höheren Geschäftswert verspricht. Extreme Programming formuliert, dass dies zu einem völlig anderen Verhalten während der Entwicklung führt. Es gilt „wichtige Entscheidungen so spät wie möglich im Entwicklungsprozess [zu][43] treffen, um die Kosten aufzuschieben, die durch diese Entscheidungen bedingt sind, und um die Wahrscheinlichkeit zu erhöhen, dass die richtigen Entscheidungen getroffen werden.“ (Bec00, S. 23) Dabei geht es nicht darum, sich um Entscheidungen zu drücken. Der Teilsatz „so spät wie möglich“ umfasst auch, dass die Entscheidung nicht zu spät gefällt werden darf.

Extreme Programming stellt eine Reihe von Techniken zur Verfügung um abzusichern, dass die These von änderbarer Software tatsächlich erfüllt werden kann.

Durch die Softwarearchitektur, erarbeitet während des Entwurfs, wird die zu entwickelnde Software strukturiert und geordnet. Eine nachträgliche Änderung kann diese Ordnung zerstören, wenn die Änderung sich mit der Softwarearchitektur nicht vollständig erfüllen lässt. Durch jede Änderung wird somit die interne Struktur der Software ungeordneter. Nur durch die bewusste Zuführung von Aufwand kann das Streben zur Unordnung verhindert werden. Extreme Programming beschreibt, welche Maßnahmen konkret ergriffen werden müssen.

### 7.2.4 Die 12 Grundpraktiken

Aufbauend auf den vier Grundwerten definiert Extreme Programming 12 Grundpraktiken (vgl. Bec00, S. 53ff), die alle umgesetzt werden müssen. Die Grundpraktiken sind sehr konkret im Vergleich zu anderen Vorgehensmodellen. Beck (vgl. Bec00) betont mehrfach, dass die Grundpraktiken bereits seit vielen Jahren bekannt sind. Allerdings ist die Kombination und konsequente (extreme) Anwendung einmalig. Die 12 Grundpraktiken lauten:

- Versionsplanung
- Kurze Releasezyklen
- Metapher für das System
- Einfaches Design
- Testen
- Refaktorisierung
- Programmieren in Paaren

[43] Anmerkung des Verfassers

- Gemeinsame Verantwortlichkeit
- Fortlaufende Integration
- 40-Stunden-Woche
- Kunde vor Ort
- Programmierstandards

Es werden nun die einzelnen Grundpraktiken näher erläutert. Dies erfolgt in einer anderen Reihenfolge als sonst in der Extreme Programming Literatur üblich, da so leichter Zusammenhänge verdeutlicht werden können.

**Kurze Releasezyklen:** Damit der Kunde möglichst schnell realisierte Funktionen einsetzen kann, werden kurze Releasezyklen angestrebt. Dabei wird von ein bis zwei Monaten als Richtwert für die Zykluslänge im Rahmen von Extreme Programming ausgegangen. Ein Release soll dabei sinnvoll sein, d. h. es können mit dem Release ein bestimmter Teil der Geschäftsanforderungen vollständig erfüllt werden. Mit späteren Releases steigt somit der Teil der erfüllten Geschäftsanforderungen. Dabei sind immer zuerst die Geschäftsanforderungen zu erfüllen, die dem Kunden den größten wirtschaftlichen Wert versprechen. Die Priorität der Geschäftsanforderungen legt der Kunde mit Unterstützung der Entwickler fest.

**Versionsplanung:** Geschäftsanforderungen werden als so genannte *Geschichten* durch den Kunden formuliert. Die Formulierung erfolgt während der Versionsplanung. Man findet für den Begriff Versionsplanung noch die Bezeichnungen *Planungsspiel* und *Releaseplanung*. Das Ergebnis der Versionsplanung ist der *Versionsplan*. Der Versionsplan legt die Zeitpunkte der einzelnen Versionen (Releases) fest. Weiterhin beschreibt der Versionsplan grob, welche Geschichten für die einzelnen Versionen erfüllt sein müssen. Da der Kunde jederzeit seine Anforderungen ändern darf, muss eine Versionsplanung bei größeren Änderungen der Anforderungen erneut durchgeführt werden. Auf den Prozess der Versionsplanung und Anforderungsverwaltung wird im Abschnitt 7.2.5 auf Seite 55 genauer eingegangen.

**Metapher für das System:** In Extreme Programming findet keine explizite Phase zur Planung der Softwarearchitektur statt. Trotzdem wird eine Metapher für das zu entwickelnde System gesucht. Diese Metapher soll zur besseren Kommunikation zwischen Entwickler und Kunde dienen. Weiterhin unterstützt die Systemmetapher die Entwicklung. Es werden z. B. Klassen entsprechend der Metapher benannt. Wahrscheinlich wurde der Begriff Metapher gewählt, um sich gegen den Begriff Architektur abzugrenzen.

Die Metapher ist eine abstrakte Beschreibung der Architektur. Sie soll allen Beteiligten „ein schlüssiges Modell zur Verfügung (...) stellen, mit dem sie arbeiten können und das sowohl für die Geschäftsleute als auch für die Techniker brauchbar ist.“ (Bec00, S. 56)

**Testen:** Der Kunde formuliert die Geschichten während der Versionsplanung und später bei Bedarf. Die Geschichten bilden die Anforderungen an das zu entwickelnde System. Zur Überprüfung, ob das System eine konkrete Anforderung erfüllt, muss der Kunde entsprechende Tests formulieren. Dies kann mit der Unterstützung der Entwickler geschehen. Diese *Funktionstests* sind das Prüfkriterium für die Erfüllung der einzelnen Geschichten.

Neben diesen Funktionstests existieren die *Komponententests*. Diese testen einzelne Klassen oder Methoden im Sinne eines Blackbox-Tests[44]. Die Komponententests werden von den Entwicklern erstellt, bevor sie mit der Programmierung der Klasse oder Methoden beginnen. Diese Herangehensweise wird in der Softwareentwicklung als test-driven (durch Test getrieben) bezeichnet. Dabei ist wichtig, dass die Tests vor der Programmierung geschrieben werden. Weiterhin sollte ein Werkzeug für die Verwaltung der Tests genutzt werden. Idealerweise führt dieses Werkzeug bei Bedarf eine Prüfung aller Tests vollautomatisch aus und gibt dem Entwickler damit eine konkrete Rückmeldung, ob alle Tests erfüllt sind. Die Tests geben dem Entwickler somit eine unmittelbare Rückmeldung, ob seine durchgeführten Änderungen unbeabsichtigte Auswirkungen haben.

Das vollautomatische Testen nach jeder Codeintegration stellt ein Mittel zur Verfügung, um die Änderbarkeit der Software zu garantieren.

**Einfaches Design:** Der Grundwert *Einfachheit* fordert, dass stets die einfachste Lösung umgesetzt werden soll. Ein einfaches Design zeichnet sich durch folgende vier Eigenschaften aus (vgl. Bec00, S. 57):

1. Es besteht alle Tests.
2. Es enthält keine Redundanzen.
3. Es setzt die Metapher um.
4. Es besteht aus der geringst möglichen Anzahl von Klassen und Methoden.

Das Design darf die Realisierung zukünftiger Anforderungen nicht vorwegnehmen, da nicht vorhergesagt werden kann, ob diese zukünftigen Anforderungen tatsächlich umgesetzt werden müssen. Der Kunde könnte z. B.

[44]Im Blackbox-Test von Klassen werden lediglich die Schnittstellen getestet. Dazu wird von der Klasse eine bestimmte Ausgabe erwartet, wenn eine bestimmte Eingabe erfolgt. Stimmt die tatsächliche Ausgabe nicht mit der erwarteten Ausgabe überein, dann ist der Test gescheitert.

eine der bereits berücksichtigten Anforderungen komplett streichen. Der bis dahin vorweggenommene Entwicklungsaufwand ist dann umsonst und hat unnötige Kosten verursacht.

Es ist zu beachten, dass es keinen expliziten Entwurfsschritt in Extreme Programming gibt. Tatsächlich legt das Schreiben von Tests vor der Programmierung das Design der zu programmierenden Klassen bzw. Methoden fest. Ein Test wird stets über die von der Klasse oder Methode zur Verfügung gestellte Schnittstelle durchgeführt. Die Beschreibung dieser Schnittstelle stellt somit den eigentlichen Entwurf der Klasse dar.

**Refaktorisierung** Jede Änderung am Code kann einen Teil der inneren Struktur zerstören. Mit dem ständigen Überarbeiten wird dem entgegengewirkt. Bei dieser Überarbeitung darf die Funktionalität des Systems nicht geändert werden. Um dies zu sichern, muss das System mit den Komponenten- und Funktionstest nach der Refaktorisierung geprüft werden. Refaktorisierung lässt sich somit definieren als „eine Änderung an der internen Struktur einer Software, um sie leichter verständlich zu machen und einfacher zu verändern, ohne ihr beobachtbares Verhalten zu ändern.“ (Fow00, S. 41)

Die Refaktorisierung muss immer dann durch die Entwickler ausgeführt werden, wenn durch eine Änderung das aktuelle Design, und damit die innere Struktur des Systems, verschlechtert wird. Mit Refaktorisierung wird die Grundlage für änderbare Software geschaffen. Durch Refaktorisierung können im wesentlichen folgende drei Ergebnisse erzielt werden (vgl. Fow00, S. 43ff):

1. Verbesserung des Design, hinsichtlich der oben genannten vier Kriterien für einfaches Design

2. Leichtere Verständlichkeit der Software

3. Erkennung von Fehlern

**Programmieren in Paaren:** Die Programmierung erfolgt in Extreme Programming paarweise. Jeder Code, der in eine Version (Release) eingeht, muss durch ein Paar erstellt werden, ansonsten muss der Code verworfen werden. Bei der Programmierung in Paaren sitzen zwei Mitarbeiter gemeinsam vor einem Rechner. Ein Mitarbeiter bedient den Rechner, indem er z. B. den Code eingibt. Der zweite Mitarbeiter überprüft zum einen den Code, analysiert auf der anderen Seite aber auch mögliche Auswirkungen der aktuellen Änderungen. Die beiden Mitarbeiter können jederzeit ihre Rolle tauschen. Die Paarbildung erfolgt durch die Mitarbeiter. Paare sollten nach der Fertigstellung einer Aufgabe gewechselt werden. Es findet somit eine ständige Fluktuation statt.

Das Programmieren in Paaren verfolgt mehrere Anliegen. So soll die Kommunikation zwischen den einzelnen Teammitgliedern gefördert werden. Weiterhin erhöht die Programmierung in Paaren die Qualität des produzierten Codes wesentlich. Durch den ständigen Austausch zwischen den Entwicklern wird ein Lernen während der Arbeitstätigkeit ermöglicht. Ein weiterer Effekt des Programmieren in Paaren ist, dass alle Mitarbeiter grob den gesamten Code kennen. Dadurch wird verhindert, dass bestimmte Codebereiche nur von einzelnen Mitarbeitern verstanden werden und deren Ausfall, z. B. bei Krankheit aber auch bei Kündigung, die Wartung bzw. Änderung dieser Codebereiche unmöglich macht. Abschließend führt die Programmierung in Paaren zu einer gegenseitigen Kontrolle sowohl inhaltlich als auch organisatorisch. Die Mitarbeiter unterlassen deshalb teilweise unproduktive Tätigkeiten wie während der Arbeitszeit im Internet zu surfen.

**Fortlaufende Integration:** Sobald ein Programmierpaar eine Aufgabe umgesetzt hat, muss es die Testfälle sowie die Änderungen am Code in die zentrale Codeverwaltung (Repository[45]) einspielen. Danach wird das Programm erneut übersetzt (kompiliert) und alle Tests werden ausgeführt. Erst wenn alle Tests vollständig bestanden sind, ist die Integration abgeschlossen und das nächste Programmierpaar darf mit der Integration beginnen. Sollte es dem Programmierpaar nicht gelingen, alle Tests zu erfüllen, muss es den Code wegwerfen und von vorne beginnen. Extreme Programming schlägt vor, dass nur ein Integrationsrechner zur Verfügung steht. Nicht integrierter Code muss am Ende des Arbeitstages verworfen werden. Durch die fortlaufende Integration soll erreicht werden, dass jederzeit eine auslieferbare Version zur Verfügung steht, die der Kunde testen kann. Als Werkzeug bietet sich hier der Einsatz einer Versionsverwaltung an.

**Gemeinsame Verantwortlichkeit:** In Extreme Programming gibt es keinen Besitz bestimmter Codebereiche. Jeder Entwickler darf an allen Stellen im Code ändern. Gleichzeitig ist jeder Entwickler für die Qualität des gesamten Codes verantwortlich. Entdeckt er in einem Bereich Schwächen, muss er diese mittels Refaktorisierung überarbeiten.

**Programmierstandards:** Damit eine gemeinsame Verantwortlichkeit realisiert werden kann, müssen sich die Entwickler auf gemeinsame Programmierstandards einigen. Dadurch wird eine gemeinsame Kommunikationsbasis geschaffen. Weiterhin soll dies verhindern, dass die Programmierer die Formatierung des Codes ändern, wenn sie an einem Bereich arbeiten.

[45] In solch einem System werden die unterschiedlichen Versionen des Codes verwaltet. Dabei können mehrere Versionen einer Datei parallel, z. B. für unterschiedliche Kundenversionen, verwaltet werden. Weiterhin kann nachvollzogen werden, wer welche Änderungen wann am Code durchgeführt hat.

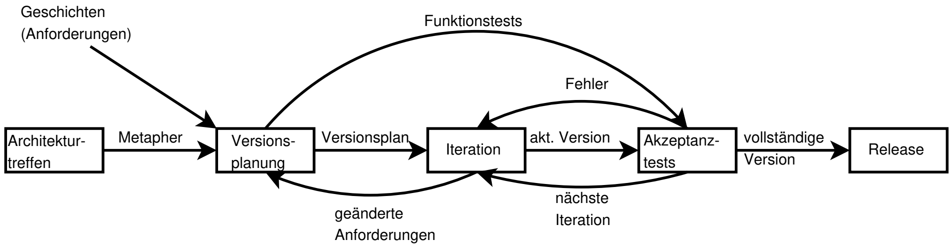


Abbildung 11: globaler Ablauf eines Extreme Programming Projektes (nach Wel99)

**Kunde vor Ort:** Der Kunde nimmt in Extreme Programming eine zentrale Rolle ein. Der Kunde wird als positiv wirkender Partner angesehen. Dabei hat der Kunde zum einen die Rolle des Auftraggebers und zum anderen die Rolle des zukünftigen Nutzers zu erfüllen. Damit der Kunde jederzeit für Rückfragen bei Problemen, Streitpunkten und für die Setzung von Prioritäten zur Verfügung steht, muss er sich möglichst im gleichen Raum wie die Entwickler befinden. Der Kunde wird dabei meist von einem zukünftigen Nutzer der Software vertreten. Diese Person muss über Entscheidungsbefugniss verfügen, damit er die oben genannten Punkte verbindlich klären kann.

**40-Stunden-Woche:** Extreme Programming legt Wert auf einen vernünftigen Umgang mit den Mitarbeitern. Diese sollen keine geplanten Überstunden leisten. Überstunden werden als Ausdruck schlechter Managementtätigkeit angesehen. Extreme Programming geht davon aus, dass Überstunden unproduktiv sind und langfristig die Produktivität der Mitarbeiter durch Demotivierung und Erschöpfung (Burnout) sinkt.

### 7.2.5 Planung und Anforderungsverwaltung

Extreme Programming wird oft vorgeworfen, dass es lediglich eine Legitimation für planloses Hacken darstellt und somit dem „Code and Fix“ Vorgehen entspricht. Es wird nun dargestellt, wie der Ablauf eines Extreme Programming Projektes gestaltet wird und wie die 12 Grundpraktiken zur Anwendung kommen. Eine grafische Veranschaulichung findet sich in Abbildung 11 auf Seite 55.

Zu Beginn eines Projektes muss dessen Umfang bestimmt werden. Dazu wird zunächst die Metapher gebildet und eine Vision für das Projekt aufgestellt. (vgl. BF01, S. 33ff) In dieser Phase, bezeichnet auch als *Erforschung* (Bec00, S. 131ff), wird untersucht, welche Technologien zur Umsetzung der Vision verwendet werden können. Es muss das Budget und die großen Aufgabenbereiche festgelegt werden. Es erfolgt eine Grobschätzung des Aufwandes für die Realisierung der großen Aufgabenbereiche. Weiterhin wird die zur Realisierung notwendige Infrastruktur, wie Testwerkzeug und Codeverwaltung, aufgebaut.

Im nächsten Schritt findet die *Versionsplanung* (vgl. BF01, S. 39ff) erstmalig statt. Aus den groben Aufgabenbereichen entwickelt der Kunde konkrete Anforderungen in Form von *Geschichten*. Der Umfang der Geschichten wird von den Entwicklern geschätzt. Eine Geschichte darf dabei zwischen ein bis drei Wochen Entwicklungszeit in Anspruch nehmen. Lässt sich eine Geschichte nicht in diesem Zeitraum realisieren, muss sie aufgeteilt werden bzw. mehrere kleinere Geschichten müssen zu einer großen Geschichte zusammengezogen werden. Die Formulierung der Geschichten erfolgt in der Sprache des Kunden. Es werden alle technischen Details ausgespart. Eine Geschichte sollte aus wenigen Sätzen bestehen. Weiterhin muss der Kunde für jede Geschichte einen Test beschreiben (*Funktionstest*), mit dem die Erfüllung der Geschichte geprüft werden kann.

Die Geschichten werden durch den Kunden nach deren *Geschäftswert* geordnet. Es findet somit das Setzen von Prioritäten statt. Dabei spielen technische Abhängigkeiten eine untergeordnete Rolle. Weiterhin werden die Termine für die Versionen bestimmt. Die Entwickler legen aus Erfahrungswerten fest, wie viel Entwicklungsarbeit zwischen zwei Versionen geleistet werden kann. Anhand dieser Zahl wird festgelegt, welche Geschichten bis zu welchen Versionen zu realisieren sind. Als Ergebnis der Versionsplanung steht am Ende der Versionsplan sowie die zu realisierenden Geschichten geordnet nach deren Priorität.

Während eine Produktversion (Release) aller ein bis zwei Monate veröffentlicht werden soll, erfolgt die Arbeit in *Iterationen*. Eine Iteration hat dabei eine Länge von ein bis drei Wochen. Zur Erreichung einer Produktversion sind mehrere Iterationen nötig. Am Anfang jeder Iteration steht die *Iterationsplanung*. (vgl. BF01, S. 85ff) Während der Iterationsplanung werden die Geschichten in *Aufgaben* aufgeteilt. Die Aufgaben stellen eine technische Spezifikation dar, die in Zusammenarbeit mit dem Kunden erarbeitet wird. Die Entwickler übernehmen die einzelnen Aufgaben und verteilen somit die Aufgaben unter sich. Jeder Entwickler schätzt, wie lange er für die Realisierung der einzelnen übernommenen Aufgaben benötigt. Die Summe der geschätzten Aufgaben stimmt nicht zwangsläufig mit der Grobschätzung für die Geschichten überein. Bei einer sich abzeichnenden Überlastung der aktuellen Iteration, erfolgt eine Rücksprache mit dem Kunden. Dieser entscheidet, welche Geschichten in eine spätere Iteration verschoben werden. Die Iterationsplanung erfolgt jeweils nur für die aktuelle Iteration. Sie stellt die Feinplanung im Rahmen von Extreme Programming dar.

Nun beginnt die eigentliche Iteration, dargestellt in Abbildung 12 auf Seite 57. Dazu sucht sich ein Programmierer für die Realisierung einer übernommenen Aufgabe einen Partner. Die Koordinierung erfolgt selbstorganisiert und nicht durch den Projektleiter. Das Programmierpaar formuliert zuerst die Tests für diese Aufgabe und bei Unklarheiten bezüglich der Auf-

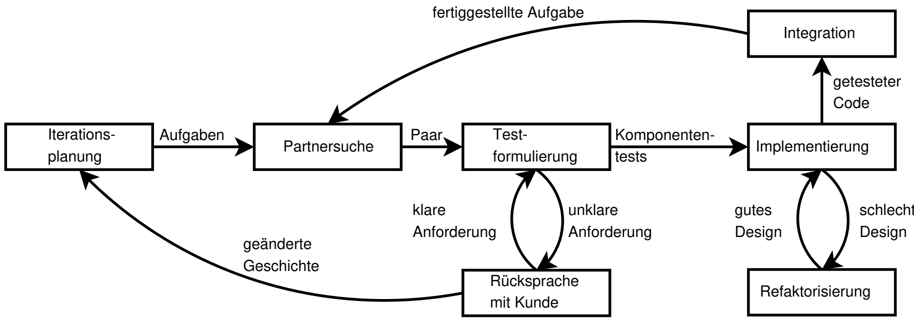


Abbildung 12: Ablauf einer Iteration in Extreme Programming (nach Wel99)

gabe findet eine Rücksprache mit dem Kunden statt. Erst dann wird der eigentliche Code durch das Programmierpaar implementiert. Dabei kann eine Überarbeitung des vorhandenen Codes nötig sein. In diesem Fall kommt die Refaktorisierung zur Anwendung. Nach wenigen Stunden wird dann der getestete Code in das Gesamtsystem integriert. Dabei wird die Erfüllung aller Tests nach der Integration sichergestellt.

Während der Iteration wird der aktuelle Fortschritt gemessen. Dadurch kann festgestellt werden, mit welcher Wahrscheinlichkeit die für die Iteration geplanten Geschichten realisiert werden können. Bei größeren Abweichungen wird eine neue Iterationsplanung durchgeführt.

Wie bereits mehrfach betont wurde, kann der Kunde jederzeit Geschichten ändern bzw. hinzufügen. Die Entwickler schätzen den Umfang der neuen bzw. geänderten Geschichte. Unter Umständen muss der Kunde entscheiden, welche Geschichten anstatt der neuen bzw. geänderten Geschichte nicht realisiert werden sollen.

### 7.2.6 Zusammenfassung Extreme Programming

Die Darstellung von Extreme Programming erfolgte sehr ausführlich um zu zeigen, dass es sich nicht um ein völlig unkontrolliertes Programmieren handelt. Es lassen sich alle Schritte der klassischen Softwareentwicklung identifizieren. Die Analyse erfolgt zum einen mit der Aufstellung der Geschichten durch den Kunden. Weiterhin werden diese Geschichten während der Iterationsplanung genauer untersucht und in konkrete Aufgaben aufgespalten. Der Entwurf erfolgt während der Programmierung. Die Architektur wird durch die Refaktorisierung schrittweise angepasst. Implementierung und Test werden eng verzahnt durchgeführt. Die Integration erfolgt täglich, die Inbetriebnahme durch den Kunden erfolgt in kurzen Zeiträumen. Die Kommunikation unter den Entwicklern wird durch das Programmieren in Paaren sowie die gemeinsame Verantwortlichkeit unterstützt. Die Kommunikation zwischen Kunde und Entwicklern wird durch die ständige Einbeziehung des Kunden, z. B. während der Versionsplanung und Iterationsplanung, gefördert. Die 12

Grundpraktiken unterstützen sich gegenseitig. (vgl. dazu Bec00, S. 63) So ist eine Refaktorisierung ohne automatische Tests nicht möglich. Eine fortlaufende Integration setzt ebenfalls die Unterstützung durch automatische Tests voraus.

Als nachteilig bei Extreme Programming ist zu bewerten, dass im Projekt gewonnenes Wissen ausschließlich intern existiert und nicht dokumentiert wird. Extreme Programming gilt als nur in kleinen Teams von etwa zehn Entwicklern umsetzbar. Für die Bewältigung größerer Projekte stehen andere agile Methodiken zur Verfügung. An dieser Stelle soll die Methodikfamilie Crystal als Beispiel vorgestellt werden.

## 7.3 Agiler Vertreter: Methodikfamilie Crystal

Während Extreme Programming konkrete Aktivitäten vorgibt, sind die Methodiken der *Methodikfamilie Crystal* wesentlich abstrakter. Crystal, entwickelt von Alistair Cockburn (vgl. Coc02; Coc03), versteht sich als eine Familie von Methodiken. Zentraler Ansatzpunkt lautet, dass Menschen und Projekte unterschiedlich sind und dass es deshalb unterschiedlicher Methodiken bedarf. Deshalb nimmt Crystal eine Klassifizierung von Projekten vor (siehe Abbildung 13[46] auf Seite 59) und bietet entsprechende Methodiken für die einzelnen Klassen an. Laut Crystal wird ein Projekt von folgenden drei Variablen charakterisiert (Coc03, S. 218):

1. Anzahl der Mitarbeiter, die koordiniert werden sollen
2. Kritizität des Projektes (Auswirkungen beim Scheitern)
3. Prioritäten des Projektes (z. B. frühzeitige Auslieferung)

Der Zusammenhang der drei Variablen lässt sich, wie in Abbildung 13 gezeigt, darstellen. Bevor für ein Projekt ein Vorgehen bestimmt werden kann, müssen zuerst die drei Variablen bestimmt werden. Weiterhin muss während des Projekt überprüft werden, ob sich die Variablen verändert haben und somit die Einstufung des Projektes verändert werden muss.

Die Begründung für dieses Vorgehen liegt in mehreren Punkten. Eine Steigerung der Mitarbeiterzahl erhöht den Kommunikationsaufwand. Auf dieses Problem hat bereits Brooks (vgl. Bro01b) aufmerksam gemacht. Dabei übersteigt ab einer bestimmten Teamgröße der Kommunikationsaufwand, verursacht durch einen weiteren Mitarbeiter, den Gewinn an Arbeitskraft durch diesen Mitarbeiter. Eine Verdoppelung der Mitarbeiter führt nicht zu einer Verdoppelung der Arbeitskraft. Cockburn (Coc03, S. 107ff) zeigt weiterhin, dass direkte Kommunikation von Mensch zu Mensch wesentlich effektiver ist, als indirekte Kommunikation. Tatsächlich ist eine direkte Kommunikation nur bei bis zu zehn Mitarbeitern im Projektteam möglich. Da

[46] Die Buchstaben (C, D, E und L) für die einzelnen Zellen stammen aus der englischsprachigen Originalliteratur.

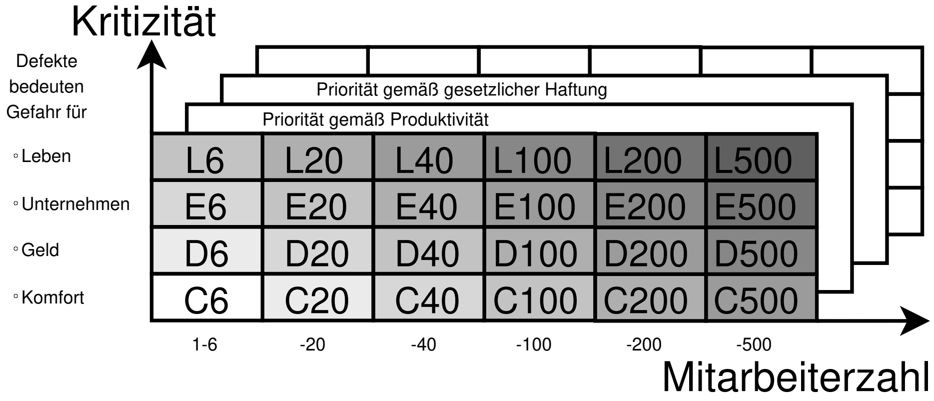


Abbildung 13: Crystal Projekteinordnung nach Anzahl Mitarbeiter, Kritizität und Prioritäten

Extreme Programming ausschließlich auf direkter Kommunikation basiert, kann es kaum in Teams größer als zehn Mitarbeiter umgesetzt werden.

Ist das Ergebnis eines Projektes kritisch für das Fortbestehen des Unternehmens, müssen andere Verfahren verwendet werden, als wenn das Projekt weniger kritisch ist. Unter Prioritäten versteht Cockburn z. B. gesetzliche Auflagen. So kann es sein, dass bestimmte Artefakte durch den Auftraggeber oder durch Randbedingungen erforderlich sind. Dann muss die verwendete Methodik die Erstellung der Artefakte sicherstellen.

In der Methodikfamilie Crystal wird davon ausgegangen, dass die Konzentration auf *Fähigkeiten der Mitarbeiter*, *Kommunikation* und *Gemeinschaft* in einem Projekt effektiver ist, als die Konzentration auf vorgeschriebene Prozesse. Daraus leiten sich folgende Grundwerte ab (Coc03, S. 267):

1. mensch- und kommunikationszentrisch

2. Möglichkeit zur Anpassung bei Bedarf

3. Toleranz

Punkt 1 bedeutet, dass Werkzeuge, Artefakte und Verfahren lediglich als Unterstützung für die menschlichen Komponente dienen. Punkt 2 stellt klar, dass die Methodik während des Projekts überarbeitet werden muss, um sich ändernden Anforderungen anzupassen. Punkt 3 hebt hervor, dass das Team anhand seiner Erfahrung entscheidet, welche Artefakte und Prozesse sinnvoll sind und damit letztendlich umgesetzt werden.

Anhand dieser Darstellung wird ersichtlich, dass die Methodikfamilie Crystal sehr abstrakt ist. Für die konkrete Umsetzung wurden je nach Einstufung des Projektes verschiedene konkrete Ausprägungen entwickelt. Die einfachste Ausprägung für kleine nicht kritische Projekte (die Zellen C6, D6 und E6 in Abbildung 13) ist *Crystal Clear*. (vgl. Coc02) Für Crystal Clear ist erforderlich (Coc02, S. 258):

- ein erfahrender Chefentwickler und zwei bis sieben weitere Entwickler
- ein großer Raum, ausgestattet mit Wandtafeln und Flipcharts
- direkte Einbeziehung des Nutzers
- regelmäßige Auslieferung von getestetem und nutzbarem Code an den Nutzer aller ein bis zwei Monate
- periodische Reflexion und Überarbeitung des Vorgehens

Dabei wird im Gegensatz zu Extreme Programming nicht auf Dokumentation verzichtet. Das Team muss allerdings selbst entscheiden, welche Artefakte sinnvoll zur Speicherung des im Projekt gewonnenen Wissens sind. Als wichtiges Werkzeug wird ein System für das Konfigurations- und Versionsmanagement angesehen. Auf den Wandtafeln und Flipcharts wird z. B. das Design und andere Entscheidungen kommuniziert. Weiterhin dienen sie während einer Diskussion im Projektteam als Zeichenfläche.

Im Vergleich zu Extreme Programming fällt auf, dass Crystal Clear weniger konkrete Forderungen stellt. Lediglich die Forderung von Dokumentation geht über den Umfang von Extreme Programming hinaus. Es werden keine Aussagen über die Programmierung (Technologie, Programmiersprache, Standards, Programmieren in Paaren, usw.) oder den Umfang von Tests getroffen. Dies liegt im Ermessen des Teams.

Für komplexere Projekte stehen eine Vielzahl von weiteren Ausprägungen der Methodikfamilie Crystal zur Verfügung. Diese sind wesentlich umfangreicher, da eine größere Anzahl von Mitarbeitern koordiniert werden muss. Auch in diesen Ausprägungen werden die drei oben genannten Grundwerte umgesetzt. Cockburn betont, dass mit zunehmender Projektgröße die Formalien zunehmen müssen, da eine direkte Kommunikation im ganzen Projektteam aufgrund des exponentiell steigenden Kommunikationsaufwandes nicht möglich ist. So bestehen z. B. bei rein direkter Kommunikation in einem Team mit sechs Mitgliedern 15 Kommunikationswege, wie in Abbildung 14 Teilbild a) auf Seite 61 nachvollzogen werden kann. In Abbildung 14 Teilbild b) wurde das Projektteam in zwei Teilteams untergliedert. Eine Kommunikation zwischen beiden Teilteams findet über jeweils eine Person aus den Teilteams statt. Durch diese Maßnahme können die Kommunikationswege mehr als halbiert werden. Allerdings wird die Kommunikation für die einzelnen Teammitglieder, wenn sie mit einem Mitglied aus dem anderen Teilteam kommunizieren müssen, erschwert, da eine Vermittlung über die Verbindungspersonen notwendig ist.

Trotzdem bleiben große formalisierte Projekte laut Cockburn agil, wenn sie die auf Seite 59 genannten drei Grundwerte umsetzen. Allgemein kann

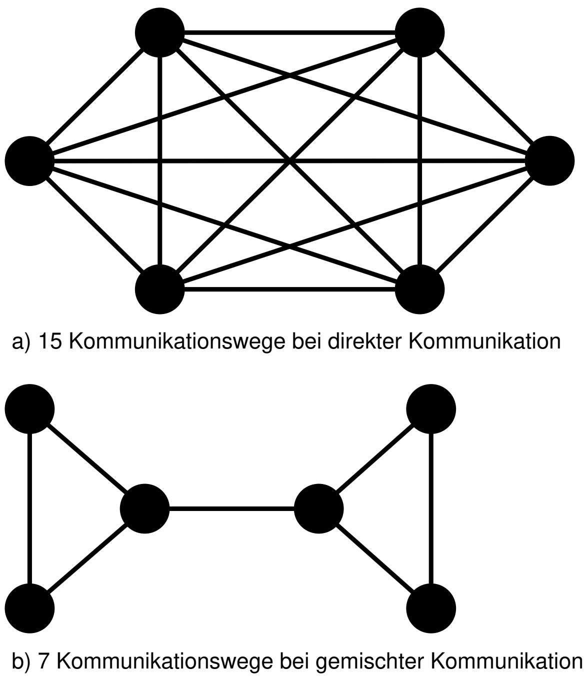


Abbildung 14: Anzahl Kommunikationswege bei direkter und bei gemischter Kommunikation am Beispiel eines Teams mit sechs Mitgliedern

davon ausgegangen werden, dass die Einführung von Formalien nicht Agilität verhindert.

## 7.4 Manifest agiler Softwareentwicklung

Es existiert eine Vielzahl von Vertretern der agilen Softwareentwicklung. In den vorherigen Abschnitten wurden

- Extreme Programming
- und Methodikfamilie Crystal

vorgestellt. Die weiteren Vertreter der agilen Softwareentwicklung sollen hier lediglich genannt werden (nach Hig02):

- Scrum
- Dynamic Systems Development Method (DSDM)
- Feature-Driven Development (FDD)
- Lean Development (LD)
- Adaptive Software Development (ASD)

Die verschiedenen Vertreter und Autoren der einzelnen Methodiken haben versucht, die Gemeinsamkeiten zu formulieren. Die entsprechend gemeinsam veröffentlichte und unterzeichnete Erklärung wird als das *Manifest agiler Software Entwicklung* (Agi01)[47] bezeichnet und lautet auszugsweise:

> „... Durch das Entwickeln von Software und indem wir anderen bei der Entwicklung helfen, erschließen wir bessere Wege der Softwareentwicklung. Durch diese Arbeit haben wir folgende Werte zu schätzen gelernt:
>
> - *Individuen und Interaktion* [zwischen den Individuen][48] sind wichtiger als Prozesse und Werkzeuge.
> - *Funktionierende Software* ist wichtiger als umfassende Dokumentation.
> - *Kundenzusammenarbeit* ist wichtiger als Vertragsverhandlungen.
> - *Auf Änderungen reagieren* ist wichtiger als einem Plan zu folgen.
>
> Wir schätzen die Punkte auf der rechten Seite, aber wir bewerten die Punkte auf der linken Seite höher. ... “

Diese vier Punkte werden durch 12 weitere Prinzipien unterstützt, auf die hier aber nicht näher eingegangen wird. Aus dem Manifest ergeben sich einige zu diskutierende Punkte. Zunächst weisen die Unterzeichner darauf hin, dass sie selber in der Softwareentwicklung tätig sind und sich ihre Ansichten daher primär aus praktischer Erfahrung ergeben haben. Weiterhin muss betont werden, dass es sich bei den vier Grundwerten nicht um ein „Entweder-Oder“ handelt. Die Unterzeichner legen lediglich eine Priorität auf die links stehenden Werte.

Die vier Punkte des Manifest lassen sich anhand der beiden in dieser Arbeit vorgestellten Vertreter der agilen Softwareentwicklung nachvollziehen. So steht die Zusammenarbeit und Kommunikation zwischen Individuen bei Crystal im Mittelpunkt des Interesses. Je nach Projektgröße werden unterschiedliche Kommunikationsformen vorgeschlagen, um die Interaktion zwischen den Projektmitgliedern zu sichern. Extreme Programming fordert die ständige Verfügbarkeit einer funktionierenden Software und verzichtet größtenteils auf Dokumentation. Sowohl bei Crystal als auch bei Extreme Programming wird eine partnerschaftliche Zusammenarbeit mit dem Kunden betont. Der Kunde sollte möglichst ins Projektteam integriert werden. Durch die iterative Überarbeitung der Planung bei Extreme Programming,

[47] Hervorhebung im Original; Übersetzung entnommen bei (Coc03, S. 281).

[48] Anmerkung des Verfassers

wird auf sich ändernde Anforderungen reagiert. Crystal schlägt die konsequente Anwendung von Reflexion während des Projektes vor und sorgt somit ebenfalls für eine Anpassung an geänderte oder neue Anforderungen.

Das Manifest agiler Softwareentwicklung ist somit eine Zusammenfassung der gemeinsamen zugrunde liegenden Prinzipien. Es klärt aber nicht das gemeinsame Weltbild[49], sondern wendet dieses konkret an. Auch wenn es sicher einer enormen Kraftanstrengung bedurfte das Manifest zu formulieren, so ist es nicht ausreichend. Das Manifest kommuniziert zwar die Grundprinzipien, benennt aber nicht direkt das gemeinsame Weltbild der agilen Softwareentwicklung. Im nun folgenden Abschnitt wird diese Grundvision der agilen Softwareentwicklung herausgearbeitet. Dabei wird zunächst der Begriff Agilität untersucht.

## 7.5 Begriff Agilität

Neben dem Manifest der agilen Softwareentwicklung muss der Begriff *Agilität* zum Verständnis der agilen Softwareentwicklung betrachtet werden. Dabei wird von einer Welt ausgegangen, die einem ständigen Wandel unterliegt. Im Rahmen der Softwareentwicklung könnte sich die Welt z. B. wandeln durch:

- Änderung im Projektteam (z. B. wechselnde Teamzusammensetzung)
- Änderung im technologischen Umfeld (z. B. Wegfall der Unterstützung für bestimmte verwendete Technologien)
- Änderung der Anforderungen durch den Kunden
- Änderung durch Konkurrenz (z. B. Veröffentlichung einer neuen Version eines Konkurrenzproduktes)
- Änderung im Markt (z. B. Fusion eines einstigen Partners mit einem Konkurrenten)
- ...

Sowohl klassische als auch agile Softwareentwicklung beschreiben Mittel, um diesen Wandel zu bewältigen. So existiert in der klassischen Softwareentwicklung z. B. das Change Management und Risikomanagement. Der agile Vertreter Extreme Programming nimmt lediglich eine kurzfristige Planung vor, um auf Änderungen in den Anforderungen reagieren zu können.

Während die klassische Softwareentwicklung Wandel als ein erfolgreich zu verwaltendes Risiko betrachtet, sieht die agile Softwareentwicklung Wandel als eine Chance. Daraus leitet sich für die klassische Softwareentwicklung die Forderung ab, Wandel möglichst zu vermeiden bzw. die Auswirkungen

[49] siehe dazu Kapitel 4 auf Seite 21

auf das Projekt so gering als möglich zu halten. Für die agile Softwareentwicklung hingegen bedeutet dies, Wandel als integralen Bestandteil eines Projektes zu akzeptieren. Dies sind zwei vollständig entgegengesetzte Positionen in Bezug auf Wandel der Welt.

Akzeptiert man eine sich wandelnde Welt, dann bedeutet dies nicht, dass jede Planung sinnlos ist. Man ist sich vielmehr bewusst, dass Pläne keine Vorwegnahme der Zukunft sind, sondern lediglich eine Hypothese der zukünftigen Entwicklung darstellen. Deshalb kann man Pläne nicht realisieren, sondern man kann lediglich nachträglich die Qualität der Hypothese prüfen. Die Zukunft einer Welt ist dann nicht vorhersagbar, wenn die Welt auf *Nicht-Linearität* und *Nicht-Determinismus* beruht. Da die agile Softwareentwicklung von solch einer sich wandelnden Welt ausgeht, bildet das *neue Weltbild*[50] die philosophische Grundlage der agilen Softwareentwicklung.

Agilität bedeutet aber nicht nur Wandel zu akzeptieren. Agilität geht einen Schritt weiter und fordert Wandel, wenn dies sinnvoll erscheint. Dies kann vorteilhaft sein, z. B. wenn man einen Wandel auslöst, dem die Konkurrenz nicht oder nur unter hohem Aufwand folgen kann. Wandel der Anforderungen kann sinnvoll sein, wenn sich durch die geänderten Anforderungen und die entsprechend realisierte Software höhere Geschäftswerte realisieren lassen.

Demnach müssen zwei Bedingungen erfüllt sein, damit eine Organisation (z. B. ein Unternehmen) als agil bezeichnet werden kann (nach Hig02, S. 5f und S. XXIII):

1. Die Organisation akzeptiert und bewältigt Wandel.

2. Die Organisation nutzt und löst Wandel zum eigenen Vorteil aus.

Das ist die Bedeutung von Agilität. Damit eine Organisation agil sein kann, muss sie *gewandt* sein (Hig02, S. 30f), d. h. sich schnell neuen Situationen anpassen können, teilweise auch durch *Improvisation*. Bevor man aber improvisieren kann, muss man zuerst das Gebiet beherrschen und die Grenzen kennen. Neben umfangreichen theoretischem Wissen setzt dies praktische Erfahrung voraus.[51] Darüber hinaus muss eine agile Organisation *flexibel* sein (Hig02, S. 33f), d. h. die Bereitschaft zur Überarbeitung bewährter Vorgehen muss existieren. Das bedeutet nicht, dass sich die Organisation mit jedem Wandel selbst ändern muss, denn eine sich ständig verändernde Struktur ist keine Organisation. Es gilt somit eine Organisationsstruktur zu finden, die situationsabhängige Anpassungen zulässt.

Für die Softwareentwicklung bedeutet dies, den durch Wandel zusätzlich verursachten Arbeitsaufwand möglichst gering zu halten. Ein Mittel ist die Reduzierung der zu ändernden Dokumentation. Weiterhin kann der Einsatz

---

[50] siehe dazu Kapitel 4.3 auf Seite 22

[51] So muss der Jazzmusiker zuerst sein Instrument und ein breites Repertoire an Stücken beherrschen, bevor er improvisieren kann.

von Werkzeugen helfen, Wandel effektiv zu bewältigen. Dabei darf aber kein Wandel ausgelöst werden, um das Vorhandensein der Werkzeuge zu rechtfertigen. Weitere konkrete Umsetzungen von Agilität finden sich bei den einzelnen Vertretern der agilen Softwareentwicklung und werden an dieser Stelle nicht diskutiert.

Vor der Formulierung des Manifest agiler Softwareentwicklung wurden die heute als agil bezeichneten Methodiken als „leicht" charakterisiert. Der Begriff „leicht" ist allerdings nicht treffend, da man in diesem Zusammenhang annehmen könnte, dass klassische Vorgehensmodelle von bestimmten Aktivitäten und Artefakten bereinigt werden und somit die agilen Methodiken lediglich eine Teilmenge der klassischen Vorgehensmodelle sind. Wie bereits an den beiden vorgestellten Vertretern der agilen Softwareentwicklung deutlich gezeigt wurde, ist dem nicht so. „Leicht" bezieht sich somit auf die Gestaltung der Softwareentwicklung. Gemeint sind damit wenig formalisierte Methodiken, die „leicht aber ausreichend" (Coc03, S. 233) sind. Tatsächlich können aber auch sehr stark formalisierte Projekte agil sein, wenn sie Wandel akzeptieren und Wandel bewusst gestalten und damit das neue Weltbild als philosophische Basis wählen.

## 7.6 Emergent Design

Im Umfeld der agilen Methodiken taucht der Begriff *Emergent Design* seit etwa Ende 2000 auf. Darunter versteht man das ohne bewusste Steuerung gestaltete Design einer Software. Emergent Design bezeichnet somit das Design als Artefakt und nicht die Designphase.

Bei Emergent Design existiert keine abgrenzbare Designphase, es findet keine bewusste Gestaltung statt. Im Rahmen von Extreme Programming werden die folgenden drei Grundpraktiken für die unbewusste Entstehung des Designs verantwortlich gemacht:

- Einfaches Design
- Refaktorisierung
- Komponententests

Mit der Konzentration auf ein *einfaches Design* während der Programmierung wird die Basis gelegt. Das einfache Design entspricht nicht dem endgültigen Design der Software, da *Refaktorisierung* für eine stetige Überarbeitung und Weiterentwicklung des Design sorgt. Dabei stellen die *Komponententests* sicher, dass das Design immer noch die volle Funktionalität unterstützt.

Der Begriff *Emergent Design* (vgl. z. B. Cav00) stammt nicht aus der Softwareentwicklung. Tatsächlich wird das Wort Emergenz in englischsprachigen Veröffentlichungen sehr häufig verwendet. In allen Bereichen des Lebens und Zusammenlebens finden täglich Gestaltungsentscheidungen statt.

So legt z. B. der Architekt das Design eines Gebäudes fest, Experten entwerfen die Grundlage einer gesellschaftlichen Institution (wie Hochschulpolitik) und Unternehmen ändern ihre interne Struktur und damit ihr Design.

Soll Emergent Design tatsächlich emergente Phänomene hervorrufen, dann muss es *Nicht-Linearität* und *Nicht-Determinismus* ermöglichen und einsetzen. Das oben beschriebene Vorgehen innerhalb von Extreme Programming ist nicht-deterministisch, da keine Vorherbestimmung der Ergebnisse (des Design) stattfindet. Es wird auf eine schrittweise Entwicklung gesetzt und eine globale Planung der Ergebnisse vermieden. Weiterhin sorgen Refaktorisierung und Programmierung für Nicht-Linearität. Es findet eine ständige Überarbeitung des aktuellen Designs der Software statt. Da es unendlich viele Möglichkeiten für das Design einer Software gibt, ist nicht vorhersehbar, welcher Entwicklungsweg eingeschlagen wird. Dies hängt, bei gleichbleibenden Projektziel, sehr stark von den Fähigkeiten und persönlichen Vorlieben der Entwickler ab. Diese vielen kleinen Änderungen führen zu einer Entwicklung, die weder vorhersehbar noch steuerbar ist.

Da scheinbar im Rahmen von Emergent Design Nicht-Linearität und Nicht-Determinismus vorliegen, kann davon ausgegangen werden, dass durch Emergent Design tatsächlich ein emergentes System gestaltet wird.

## 7.7 Einordnung in die Systematik

Die meisten der vorgestellten Techniken dienen der *Bewältigung* eines emergenten Systems. Sie helfen den Wandel und die sich ändernden Anforderungen zu bewältigen. Durch kurze Releasezyklen und die Iterationsplanung können neue Anforderungen beachtet werden und die automatisierten Tests und die fortlaufende Integration stellen die Funktionalität der Software nach Änderungen sicher. Hier findet ein Umgang mit den beiden Prinzipien Nicht-Linearität und Nicht-Determinismus statt.

Die enge Vernetzung der Mitarbeiter sowie das Emergent Design stellen eine mögliche *Gestaltung* eines emergenten Systems dar. In der Zukunft werden sich mit Sicherheit weitere Anwendungen ergeben. Hier findet eine Nutzung der beiden Prinzipien Nicht-Linearität und Nicht-Determinismus statt.

## 7.8 Zusammenfassung agile Softwareentwicklung

In diesem Abschnitt wurden zwei Vertreter der *agilen Softwareentwicklung* vorgestellt. Anhand dieser Beispiele wurden die Gemeinsamkeiten der agilen Methodiken herausgearbeitet. Diese Gemeinsamkeiten wurden im *Manifest der agilen Softwareentwicklung* zusammengefasst. Mit der Diskussion des Begriffs *Agilität* konnte gezeigt werden, dass das *neue Weltbild* die philosophische Grundlage der agilen Softwareentwicklung ist. Weiterhin wurde das Phänomen *Emergent Design* als eine mögliche Gestaltung eines emergenten Systems vorgestellt. Abschließend wurden einzelne Verfahren der agilen Softwareentwicklung in die entwickelte Systematik eingeordnet.

Im nun folgendem Abschnitt wird die eigentliche Fragestellung dieser Arbeit beantwortet.

# 8 Emergenz in der Softwareentwicklung

## 8.1 Einleitung

Der Gegenstand dieser Arbeit ist die Fragestellung, ob in der Softwareentwicklung Emergenz angewendet werden kann und ob dies bereits geschieht. In diesem Kapitel wird eine umfassende Antwort, aufbauend auf die in den vorhergehenden Kapiteln entwickelten Darstellungen, gegeben. Entsprechend der in Kapitel 4.4 auf Seite 24 entwickelten Systematik, findet eine Unterscheidung in *Bewältigung* und *Gestaltung* einer emergenten Welt statt. Aus dieser Betrachtung kann die Antwort auf die Fragestellung dieser Arbeit formuliert werden.

## 8.2 Bewältigung von Emergenz in der Softwareentwicklung

Unter Bewältigung von Emergenz versteht diese Arbeit den Umgang mit einer Welt geprägt durch *Nicht-Linearität* und *Nicht-Determinismus*. Bevor mit einer Bewältigung begonnen werden kann, muss allerdings zunächst die Einsicht in die Notwendigkeit existieren. Dies bedeutet eine Abkehr von der These, wonach Softwareentwicklung eine reine *Ingenieurdisziplin* ist – eine Abkehr vom *mechanischen Weltbild*. Diese Abkehr vom mechanischen Weltbild hin zum *neuen Weltbild* mündet in der Kritik an der *Baumetapher* der klassischen Softwareentwicklung:

- Software ist leichter änderbar
- Software ist Teil einer Problemlösung und kein reines Produkt
- Software kann nicht endgültig fertig gestellt werden, sondern es erfolgt eine stetige Weiterentwicklung
- Software ist abstrakt und damit schlecht vermittelbar
- Entwicklung ist nicht langfristig planbar, da Anforderungen und Umwelt sich während des Projektes ändern
- Parallelität der Phasen Analyse, Entwurf, Implementierung, Test und Inbetriebnahme ist während Softwareentwicklung möglich
- ...

Mit Sicherheit ließen sich noch weitere Kritikpunkte finden, aber schon diese wenigen Punkte sind ausreichend um zu erkennen, dass eine Baumetapher für die Softwareentwicklung nicht ausreichend ist. Zentraler Kritikpunkt an der Baumetapher ist, dass Software im Vergleich zu einem Ingenieurprodukt wie einer Brücke oder einem Haus jederzeit änderbar ist. Ein Haus kann nur unter hohem Kostenaufwand geändert werden, wenn es sich bereits im Rohbau oder sogar im Innenausbau befindet. Daraus leitet sich die

Forderung ab, dass vor Baubeginn alle zukünftigen Anforderungen ermittelt und berücksichtigt sein müssen. Software hingegen kann jederzeit geändert werden. Der dafür notwendige Kostenaufwand kann durch die vorgestellten Techniken wie *automatisierte Tests*, *Refaktorisierung* und *häufige Integration* wesentlich reduziert werden. Es gibt somit keinen Grund, warum man das Mittel der Softwareänderung nicht einsetzen sollte. Tatsächlich ist es das Mittel, um in einer dynamischen Welt erfolgreich Software zu entwickeln. Denn Anforderungen ändern sich. Es ist unmöglich alle Anforderungen in einem komplexen Projekt am Projektanfang in einer Analysephase vollständig zu bestimmen. Bereits kleinste Änderungen der Anforderungen können große Auswirkungen auf die zu entwickelnde Software haben.

Der typische Einsatz von Software erfordert eine häufige Überarbeitung und Weiterentwicklung. Im Gegensatz zu dem Produkt eines Ingenieurprozesses wird Software oft erweitert oder Kernfunktionen müssen an neue Rahmenbedingungen angepasst werden. All dies erfordert eine Änderung der Software nach deren ursprünglicher Fertigstellung.

Es ist bekannt, dass Anforderungen nur schwer ermittelt werden können. Software ist letztendlich der abstrakte Teil einer komplexen *Problemlösung*. Dem Nutzer fällt es dementsprechend schwer, die Anforderungen an diese abstrakte Komponente genau zu spezifizieren. Dem wurde in der Vergangenheit durch die Entwicklung von Prototypen und neuen Analysetechniken entgegengewirkt. Die beste Möglichkeit die während der Analyse gewonnenen Anforderungen zu überprüfen, ist der Einsatz des Produktes in der späteren Produktionsumgebung. Dies bedeutet eine häufige Auslieferung von Softwareversionen, die der Nutzer prüft, damit er seine Anforderungen überarbeiten und präzisieren kann. Es handelt sich hierbei um eine iterativ inkrementelle Entwicklung mit stark verkürzten Zyklen von wenigen Monaten. Dabei ergeben sich mehrere Vorteile. So bringt jede neue Version lediglich eine geringe Anzahl von Neuerungen. Damit wird der Nutzer zum stetigen Lernen aufgefordert, aber es findet keine Überforderung der Lernfähigkeit statt. Eine Überforderung kann dann eintreten, wenn zu einem Zeitpunkt z. B. eine komplette Software eingeführt werden soll und wesentliche Teile der Arbeitsprozesse davon betroffen sind und neu erlernt werden müssen. Weiterhin ist es zum Vorteil des Kunden, wenn er während des Projektes seine Anforderungen überarbeiten kann. Dadurch erhält er eine Software, die seinen tatsächlichen aktuellen Anforderungen, und nicht den Anforderungen zu Projektbeginn, entspricht. Der Kunde ist somit an der Gestaltung der zu entwickelnden Software direkt beteiligt.

Im Gegensatz zum Produkt eines Ingenieurprozesses, wie ein Automobil oder ein Haus, ist Software meist Bestandteil einer umfassenden Problemlösung. Diese Problemlösung umfasst neben dem Softwareprodukt weiterhin Hardware, Beratung, Schulung und sogar eine Überarbeitung der Geschäftsprozesse des Kunden (Business Reengineering).

Durch den Wandel der Anforderungen und eingesetzten Technologien wird eine langfristige Projektplanung erschwert. Sinnvoller ist deshalb eine zuverlässige Planung in kurzen Zeiträumen.

Verzichtet man in der Softwareentwicklung auf einen starren Ablauf der einzelnen Phasen, kann man die Entwicklung durch Parallelisierung der Phasen wesentlich beschleunigen. Während bereits Funktionen als Code implementiert werden, findet die Analyse später benötigter Funktionalität statt. Dies führt zu einer gleichmäßigeren Auslastung der einzelnen Mitglieder des Projektteams. Weiterhin können wichtige Programmfunktionen bereits sehr frühzeitig durch den Kunden eingesetzt werden.

Man erkennt, dass die Softwareentwicklung, verstanden als reine Ingenieurdisziplin, nicht alle Fragen beantworten kann. Tatsächlich versucht die klassische Softwareentwicklung Probleme wie sich ändernde Anforderungen zum Beispiel mit dem Change Management zu behandeln. Es muss dabei allerdings gefragt werden, ob mit solchen Verfahren letztendlich nicht nur die Symptome bekämpft werden, ohne die eigentliche Ursache zu erkennen. Denn die Ursache ist eine Welt geprägt von Nicht-Linearität und Nicht-Determinismus. Einige Gründe für Nicht-Linearität in der Softwareentwicklung sind:

- Menschen handeln nicht-linear, sie sind keine Maschinen
- Software ist das Produkt eines Teams von Individuen
- Code ist immer stark vernetzt und rückgekoppelt
- Softwareprodukt wird in eine komplexe Umgebung integriert
- ...

Software wird von Menschen entwickelt. Menschen handeln nicht vollständig rational, sondern sind geprägt durch eine Vielzahl von Konflikten. Das Verhalten eines Menschen ist situationsabhängig. Der Mensch denkt assoziativ und nicht rein logisch wie ein Digitalrechner. Dies führt zu einer gewissen Sprunghaftigkeit, die einzig durch den zentralen Charakter des Individuums geglättet wird. Tatsächlich ist eine vollständige Unterdrückung der Sprunghaftigkeit nicht erwünscht, da die Sprunghaftigkeit im Verhalten die Voraussetzung für Kreativität ist. Weiterhin steht der Mitarbeiter in ständiger Interaktion mit den Mitarbeitern, dem Kunden und dem Management. Diese Gruppen haben oft widersprechende Anforderungen. Die Vernetzung der Individuen führt zu einer nicht-linearen Gruppendynamik.

Jeder Codebereich, wie eine Klasse oder Methode, steht mit einer Vielzahl anderer Codebereiche in Verbindung. Eine gute Softwarearchitektur kann dieses Problem einschränken, dennoch kann sie die entstehenden Rückkoppelungseffekte durch die Vernetzung des Codes nicht vollständig verhindern, da der Code zur Erfüllung der Softwarefunktionalität zu einem Mindestmaß vernetzt sein muss. Deshalb können bereits kleinste Änderungen Auswirkungen auf das Gesamtverhalten des Systems haben.

Software wird bei der Auslieferung in eine komplexe Umgebung über eine Vielzahl von Schnittstellen integriert. Durch diese enge Koppelung der Software an die Umwelt wird die Software ein Bestandteil eines größeren Systems und verliert dadurch ihre Autonomie. Die Software ist danach abhängig von einer Vielzahl von Elementen, die nicht kontrolliert werden können. Sie ist Bestandteil eines Netzwerkes.

Weiterhin ist die Softwareentwicklung bestimmt von Nicht-Determinismus. Dies äußert sich z. B. in folgenden Punkten:

- es existiert nicht die ideale Softwarearchitektur
- zukünftige Anforderungen und Umweltbedingungen sind nicht vorhersehbar
- Technologie- und Umweltentwicklung können nicht gesteuert werden
- Planung liefert eine Hypothese über die Zukunft
- Schätzungen sind Schätzungen und keine Vorwegnahme der Zukunft
- Projekte erbringen individuelle nur bedingt vergleichbare Lösungen
- Projekte beinhalten Risiken (z. B. zu Scheitern)
- Organisationsstrukturen und Strategien sind einzigartig
- ...

Die Gestaltungsmöglichkeiten von Software sind vielfältig. Dies belegt allein die unüberschaubare Anzahl von Programmiersprachen. Weiterhin existiert eine Vielzahl von Entwicklungsparadigmen[52]. Aus dem verwendeten Paradigma leitet sich die Architektur ab. Diese Architektur kann im Konkreten ebenfalls vielfältig erfolgen. Wird z. B. eine Software in Hinblick auf Erweiterungsfähigkeit entwickelt, dann kann dies über mehrere Wege erfolgen. Denkbar ist beispielsweise eine Plugin-Architektur mit öffentlichen Schnittstellen, die das dynamische Nachladen von Funktionalität erlaubt. Eine andere Lösung ist die Integration einer Skriptsprache mit einer entsprechenden Laufzeitumgebung, damit Erweiterung direkt die internen Daten manipulieren können.

In der Softwareentwicklung sind zukünftige Anforderungen und Umweltbedingungen nicht absehbar. So können sich die Anforderung des Kunden z. B. durch Marktverschiebungen drastisch ändern. Einen ähnlich starken Einfluss haben Änderungen der verwendeten Technologie. In beiden Fällen

[52] z. B. Objektorientierung, Strukturierung, logische Programmierung, Aspektorientierung, usw.

ist der Einfluss und die Steuerungsmöglichkeit durch das Projektteam relativ gering. Durch diese Unfähigkeit die Zukunft vorherzusagen oder sogar vorherzubestimmen, ist jede Planung lediglich eine Hypothese über die Zukunft. Pläne können nicht realisiert werden, sondern es kann lediglich überprüft werden, ob die Planung der tatsächlichen Entwicklung entsprach. Es muss dabei daran erinnert werden, dass Schätzungen immer nur mit einer bestimmten Wahrscheinlichkeit richtig sind. Umso kürzer und zeitlich nah der von einer Schätzung abgedeckte Zeitraum ist, umso höher ist die Wahrscheinlichkeit einer richtigen Schätzung.

Der Vergleich mit anderen Projekten ist oft wenig hilfreich. Schon kleine Änderungen der Bedingungen, z. B. in der Zusammensetzung des Projektteams, können durch die Vernetzung und Rückkoppelung riesige Auswirkungen auf den Projektverlauf haben, was einen Vergleich unmöglich macht.

Letztendlich sind Organisationen immer einzigartig. Formal gesehen können Organisationen die selbe Struktur aufweisen, dennoch besteht eine Organisation nicht nur aus Struktur, sondern auch aus einzigartigen Individuen. Dadurch bildet jede Organisation eine eigene Kultur, die es zu berücksichtigen gilt.

Das primäre Ziel der Softwareentwicklung ist es, unter den oben genannten Bedingungen Software mit entsprechender Qualität, im gegebenen Zeitrahmen und unter Einhaltung des Projektbudgets zu entwickeln. Das sekundäre Ziel lautet, zukünftige Entwicklungen vorzubereiten (vgl. Coc03, S. 51ff). Dies kann z. B. durch Wiederverwendung bereits entwickelter Komponenten, wartungsfreundliche Gestaltung der Software usw. geschehen. Trotzdem ist eine gut wartbare Software nutzlos, wenn sie vom Kunden als nicht ausreichend betrachtet wird und somit das primäre Ziel eine die Anforderungen erfüllenden Software verfehlt wurde.

Die meisten heute bekannten Mittel zur Erreichung der beiden Ziele in einer emergenten Welt wurden bereits im letzten Kapitel und im Kapitel 5 auf Seite 25 aufgezeigt. Deshalb seien sie lediglich auszugsweise an dieser Stelle kurz genannt:

- Management gibt Rahmen vor und sorgt für die nötigen Ressourcen
- Entscheidungen werden primär auf der Ebene der Wertschöpfung getroffen
- Abflachung der Hierarchie
- Rückkoppelung im Projektteam durch direkte Kommunikation und Kommunikationsmittel
- Kunde wird als Partner verstanden
- Kunde gestaltet Produkt aktiv mit

- partnerschaftliches Verhältnis zu externen Anbietern (z. B. für Softwarekomponenten oder Beratung)
- inkrementelle Entwicklung mit kurzen Releasezyklen zur Sicherstellung von Rückmeldung durch Kunden
- Aufwandsschätzungen und Planung für kurze Zeiträume
- Optimierung von Fähigkeiten und nicht von Prozessen
- Anpassung des Vorgehensmodell an das Projekt und nicht umgekehrt
- breite Nutzung von Werkzeugen zur Unterstützung, aber nicht Werkzeug im Zentrum der Aufmerksamkeit
- häufige Integration, automatisierte Tests, Refaktorisierung, Programmierung in Paaren, Programmierstandards, usw.
- Implementierung einer lernenden Organisation, z. B. über Reflexion und Paarprogrammierung
- Messung Projekterfolg an ausgeliefertem Produkt und nicht an Einhaltung von formalen Prozessen
- Mitarbeiter sind zu einem Zeitpunkt lediglich an einem Projekt beteiligt, damit sie den größtmöglichen Anteil am Projektwissen erlangen können

Es existieren Berichte von Unternehmen wie Microsoft (vgl. CS96), die entsprechende Verfahren in der Softwareentwicklung bereits seit Jahren erfolgreich einsetzen. Ein weiteres praktisches Beispiel für erfolgreiche Bewältigung von Emergenz ist die „OpenSource“ Bewegung, die Software vollständig *selbstorganisiert* entwickelt.

Im Hinblick auf die Fragestellung dieser Arbeit kann davon ausgegangen werden, dass bereits heute eine bewusste *Bewältigung von Emergenz* stattfindet. Ob dies allerdings mit der Kenntnis des neuen Weltbildes stattfindet, erscheint fraglich. Eine Verbesserung und Erweiterung der verwendeten Mittel in Zukunft ist sehr wahrscheinlich. Gerade im Wirtschaftsbereich wurden bereits einige Mittel langfristig erprobt und kommen erst jetzt in der Softwareentwicklung zum Einsatz. Interessant ist z. B. die Fragestellung, wie das Projektwissen möglichst ohne aufwendige Dokumentationserstellung gesichert werden kann.

## 8.3 Gestaltung von Emergenz in der Softwareentwicklung

Ebenso interessant wie die Frage nach Bewältigung von Emergenz ist die Aufgabe Emergenz bewusst zu gestalten. Während im vorherigen Punkt eine lange Liste von Mittel aufgeführt werden konnte, ist dies für die Gestaltung von Emergenz in der Softwareentwicklung (noch) nicht möglich. Die Gestaltung setzt voraus, dass ein System geschaffen wird, in dem *Nicht-Linearität* und *Nicht-Determinismus* möglich sind, damit *Selbstorganisation* auftreten kann. Eine direkte Anwendung findet momentan lediglich durch *Emergent Design*[53] statt. Hier erfolgt die Entstehung einer Softwarearchitektur durch die Anwendung eines dynamischen Prozesses. Es findet keine präzise Planung statt. Überarbeitungen sind erlaubt und werden direkt gefordert.

Es kann davon ausgegangen werden, dass die Mittel zur Bewältigung von Emergenz selbst in ihrer Gesamtheit Emergenz hervorrufen. „Die Verfahren und Prinzipien [von Extreme Programming][54] arbeiten zusammen und unterstützen einander gegenseitig, um eine Synergie zu erzeugen, die größer als die Summe der einzelnen Teile ist.“ (Bec00, S. 149) Dieser Punkt bedarf zukünftig weiterer Untersuchungen.

Wahrscheinlich existieren viele weitere Verfahren und Möglichkeiten zur Gestaltung von Emergenz in der Softwareentwicklung. So ist es durchaus denkbar das in der *Sozionik* entwickelte *Agentenkonzept* in der Softwareentwicklung umzusetzen. In dieser Vision sind autonome Agenten für das Design, die Implementierung, den Test und die Dokumentation einer Software verantwortlich. Lediglich die Analyse ist durch den Menschen durchzuführen und die daraus abgeleiteten Anforderungen den Agenten mitzuteilen. Dabei ist es vorstellbar, dass für verschiedene Teilbereiche der zu erstellenden Software verschiedene spezialisierte Agenten eingesetzt werden. Anhand der den Agenten übergebenen Anforderungen stellen diese Hypothesen für die zu entwickelnde Software auf. Eine Überprüfung der Hypothesen durch den Entwickler oder – idealerweise – durch den zukünftigen Nutzer, ist sinnvoll.

Dieser Ansatz der Softwareentwicklung beruht somit auf der Verarbeitung von Wissen. Dabei geht der Ansatz aber über die Nutzung einer logischen Programmiersprache wie Prolog oder Lisp hinaus, da explizit keine Programmierung durch den Menschen erfolgt. Auf der anderen Seite kann dieser Ansatz gegen Verfahren wie Model Driven Architecture abgegrenzt werden, da bei dieser Vision kein Design durch den Nutzer erfolgt, sondern lediglich die Analyse der Nutzeranforderungen. Das Design und die darauf folgenden Arbeitsergebnisse entstehen durch sich selbstorganisierende Agenten. Diese lassen sich somit nicht auf „intelligente Quelltextgeneratoren“ in ihrer Funktion reduzieren, da sie vor der eigentlichen Quelltextgenerierung zunächst ein Design entwickeln. Bereits heute existieren einige grundlegende Forschungsarbeiten zu dieser Vision, wie z. B. das Tropos Projekt (vgl. z. B. GPS02)[55]. Neben der Sozionik könnten die Erfahrungen aus der Künstlichen Intelligenz Forschung im Bereich Wissensrepräsentation (vgl. Gör00, S. 153ff) genutzt werden.

[53] ausführlich dazu in Kapitel 7.6 auf Seite 65
[54] Anmerkung des Verfassers
[55] weitere Informationen auch online verfügbar unter http://www.troposproject.org/

Im Bereich *Gestaltung von Emergenz* in der Softwareentwicklung sollten deshalb zukünftige Forschungsarbeiten ansetzen. Für die Beantwortung der Fragestellung dieser Arbeit bedeutet dies, dass im Bereich bewusster Gestaltung und Nutzung von Emergenz in der Softwareentwicklung noch gewaltige Chancen stecken und erst eine kleine Anwendung dieser Möglichkeit stattfindet.

## 8.4 Zusammenfassung Emergenz in der Softwareentwicklung

In diesem Kapitel konnte die Fragestellung dieser Arbeit beantwortet werden. Dazu erfolgte die Beantwortung jeweils getrennt für die beiden in der Systematik entwickelten Teilbereiche.[56] Im nun abschließenden Kapitel wird die Argumentation dieser Arbeit kompakt zusammengefasst.

[56] Die Systematik wurde in Kapitel 4.4 auf Seite 24 eingeführt.

# 9 Zusammenfassung

Diese Arbeit beantwortet die Frage, ob *Emergenz* in der *Softwareentwicklung* bereits eingesetzt wird oder ob der zukünftige Einsatz ein Potenzial zur Produktivitäts- und Qualitätssteigerung darstellt.

Deshalb wurde zunächst die Bedeutung des Begriffes Emergenz ermittelt und ein Modell zum Verständnis von Emergenz als *Transformationsprozess* erarbeitet. Darauf aufbauend wurde untersucht, welche *Erklärungsmodelle* bereits heute für Emergenz existieren. Dabei konnte festgestellt werden, dass die verschiedenen Erklärungsmodelle *Nicht-Linearität* und *Nicht-Determinismus* beschreiben. Es wurde eine Arbeitsdefinition für die Begriffe gefunden. Das Zusammenspiel von Nicht-Linearität und Nicht-Determinismus ist die Grundlage von *Selbstorganisation*. Selbstorganisation wird im allgemeinen Sprachgebrauch als die Erscheinungsform von Emergenz betrachtet.

Bei der genaueren Untersuchung der beiden Begriffe Nicht-Linearität und Nicht-Determinismus zeigte sich, dass diese die philosophische Basis eines Weltbildes sind, das sich vom allgemein anerkannten *mechanischen Weltbild* nach Newton unterscheidet. Durch den Vergleich des mechanischen Weltbildes und des *neuen Weltbildes*, konnten die unterschiedlichen Grundansichten verdeutlicht werden. Demnach steht das mechanische Weltbild für eine prinzipiell berechenbare Welt, geprägt von *Linearität* und *Determinismus*.

Für die weitere Untersuchungen im Rahmen dieser Arbeit wurde die Gültigkeit des neuen Weltbild als philosophische Grundlage gewählt. Dabei wurde eine Welt nach dem neuen Weltbild als *emergente Welt* bzw. als *Emergenz in der Softwareentwicklung* bezeichnet. Um die weiteren Betrachtungen zu systematisieren, wurde zwischen einer reinen *Bewältigung* und einer bewussten *Gestaltung* der emergenten Welt unterschieden.

Anschließend wurde untersucht, welche Konzepte und Theorien zur *Bewältigung* und *Gestaltung* einer *emergenten Welt* in der Wirtschaftsinformatik existieren. Die identifizierten Vertreter wurden der Systematik entsprechend eingeordnet.

Bevor eine Betrachtung von Emergenz in der Softwareentwicklung erfolgen konnte, wurde zunächst die *klassische Softwareentwicklung* dargestellt. Dabei konnte gezeigt werden, dass die klassische Softwareentwicklung vom mechanischen Weltbild nach Newton geprägt ist. Dies äußert sich in der Betrachtung von Softwareentwicklung als *Ingenieurdisziplin*.

Als Gegenvorschlag zur klassischen Softwareentwicklung wurde die *agile Softwareentwicklung* mit zwei konkreten Vertretern vorgestellt. Dabei wurden die Gemeinsamkeiten der agilen Vertreter untersucht. Die philosophische Basis der agilen Softwareentwicklung ist das *neue Weltbild*. Dementsprechend erfolgte eine Einordnung der in der agilen Softwareentwicklung aufgezeigten Mittel und Verfahren in die Systematik. Die meisten Mittel und Verfahren dienen aber lediglich einer *Bewältigung von Emergenz in der Softwareentwicklung*.

Abschließend erfolgte die Beantwortung der Fragestellung dieser Arbeit. Dabei war eine Differenzierung der Antwort gemäß der entwickelten Systematik notwendig. Demnach findet bereits heute eine *Bewältigung von Emergenz in der Softwareentwicklung* statt. Hierbei handelt es sich primär um eine Optimierung und Weiterentwicklung der bereits bekannten Verfahren und Mittel. Im zweiten Bereich *Gestaltung von Emergenz in der Softwareentwicklung* ließen sich kaum angewandte Konzepte identifizieren. Hier besteht großer Forschungsbedarf und es bieten sich große Chancen für die Softwareentwicklung.

# Literatur

[Agi01] AgileAlliance: Das Manifest der agilen Softwareentwicklung, 2001. – online verfügbar unter http://www.agilemanifesto.org/

[Art96] Arthur, W. B.: Increasing Returns and the New World of Business. In: *Harvard Business Review* (1996), Nr. July-August, S. 100–109. – online verfügbar unter http://www.santafe.edu/arthur/

[Bec00] Beck, Kent: *Extreme Programming: Das Manifest.* München : Addison Wesley Longman Inc., 2000

[BF01] Beck, Kent ; Fowler, Martin: *Extreme Programming planen.* München : Addison Wesley Longman Inc., 2001

[Bra93] Braitenberg, Valentin: *Vehikel - Experimente mit kybernetischen Wesen.* Reinbek bei Hamburg : Rowohlt Taschenbuch Verlag GmbH, 1993

[Bro01a] Kap. No Silver Bullet - Essence and Accident in Software Engineering In: Brooks, Frederick P.: *The Mythical Man-Month.* 15., Aufl. New York : Addison Wesley Longman Inc., 2001, S. 177–203

[Bro01b] Kap. The Mythical Man-Month In: Brooks, Frederick P.: *The Mythical Man-Month.* 15., Aufl. New York : Addison Wesley Longman Inc., 2001, S. 13–28

[Bul96] Bullinger, Hans-Jörg ; Warnecke, Hans-Jürgen (Hrsg.): *Neue Organisationsformen im Unternehmen: ein Handbuch für das moderne Management.* Berlin : Springer, 1996

[Büh87] Bühl, Walter L.: Grenzen der Autopoiesis. In: *Kölner Zeitschrift für Soziologie und Sozialpsychologie* (1987), Nr. 39, S. 225–254. – online verfügbar unter http://www.vordenker.de/buehl/wlb_grenzen-autopoiesis.pdf

[Cav00] Cavallo, David P.: *Technological Fluency and the Art of Motorcycle Maintenance: Emergent Design of Learning Environments*, Massachusetts Institute of Technology, Diss., 2000. – online verfügbar unter http://web.media.mit.edu/~cavallo/

[Coc02] Cockburn, Alistair: *Crystal Clear.* 2002. – auch online verfügbar unter http://alistair.cockburn.us/crystal/books/cc/crystalclear.zip

[Coc03] Cockburn, Alistair: *Agile Software-Entwicklung.* Bonn : mitp-Verlag, 2003

[Col03] Coldewey, Jens: Änderbare Software: Was Softwareentwicklung mit der Thermodynamik verbindet. In: *Objekt Spektrum* (2003), Nr. 1, S. 26–30. – online verfügbar unter http://www.coldewey.com/publikationen/Kolumne.html

[CS96] Cusumano, Michael A. ; Selby, Richard W.: *Die Microsoft-Methode: sieben Prinzipien, wie man ein Unternehmen an die Weltspitze bringt.* Freiburg i. Br. : Rudolf Haufe Verlag, 1996

[Dud00] Duden: *Duden, Das große Fremdwörterbuch.* 2., neu bearb., erw. u. aktualisierte Aufl. Hannover : Bibliographisches Institut & F. A. Brockhaus AG, 2000

[Ess00] Esser, Hartmut: *Soziologie Spezielle Grundlagen.* Bd. 2: *Die Konstruktion der Gesellschaft*. 1. Aufl. Frankfurt/Main : Campus Verlag GmbH, 2000

[Fis93] Fischer, Hans R.: Murphys Geist oder die glücklich abhanden gekommene Welt. Zur Einführung in die Theorie autopoietischer Systeme. In: *Autopoiesis.* 2., korr. Aufl. Heidelberg : Carl-Auer-Systeme Verlag, 1993, S. 9–37

[Fli99] Fliedner, Dietrich: *Komplexität und Emergenz in Gesellschaft und Natur.* Frankfurt/Main; Berlin; Bern; Brüssel; New York : Europäischer Verlag der Wissenschaften Peter Lang GmbH, 1999

[Flä98] Flämig, Michael: *Naturwissenschaftliche Weltbilder in Managementtheorien.* Frankfurt/Main; New York : Campus Verlag, 1998

[Fow00] Fowler, Martin: *Refactoring: Wie Sie das Design vorhandener Software verbessern.* New York : Addison Wesley Longman Inc., 2000

[Gam96] Gamma, Erich: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software.* New York : Addison Wesley Longman Inc., 1996

[Geo13] Georges, Karl E.: *Lateinisch-Deutsches Handwörterbuch.* Hannover und Leipzig : Hahnsche Buchhandlung, 1913

[GF03] Gehle, Michael ; Feldhoff, Ellen: Quo vadis Knowledge Management? In: *Zukunft des Managements: Perspektiven für die Unternehmensführung.* Zürich : vdf Hochschulverlag AG, 2003, S. 165–176

[GI03] GI: Organic Computing / VDE, ITG, GI. 2003. – Positionspapier. online verfügbar unter http://www.gi-ev.de/informatik/presse/presse_030710.shtml

[GPS02] GIUNCHIGLIA, Fausto ; PERINI, Anna ; SANNICOLÒ, Fabrizio: Knowledge Level Software Engineering. In: MEYER, John-Jules C. (Hrsg.) ; TAMBE, Milind (Hrsg.): *Intelligent agents VIII: agent theories, architectures, and languages* Bd. LNAI 2333. Berlin : Springer, 2002, S. 6–20

[GS02] GORGES-SCHLEUTER, Martina: Evolutionäre Algorithmen - Vorbild Natur. In: KELLER, Hubert B. (Hrsg.): *Maschinelle Intelligenz*. Braunschweig; Wiesbaden : Vieweg & Sohn Verlagsgesellschaft mbH, 2002, S. 243–280

[Gör00] GÖRZ, Günther (Hrsg.): *Handbuch der Künstlichen Intelligenz*. 3., vollst. überarb. Aufl. München : Oldenbourg Wissenschaftsverlag GmbH, 2000

[Hak90] HAKEN, Hermann: *Synergetik: eine Einführung*. 3., erw. Aufl. Berlin : Springer, 1990

[Hak91] HAKEN, Hermann: *Synergetik: Die Lehre vom Zusammenwirken*. 2. Aufl. Frankfurt/Main; Berlin : Verlag Ullstein GmbH, 1991

[Hei86] HEISENBERG, Werner: *Der Teil und das Ganze*. 6., unv. Aufl. München : R. Piper & Co. Verlag, 1986

[Hig02] HIGHSMITH, Jim: *Agile Software Development Ecosystems*. Boston : Addison Wesley Longman Inc., 2002

[HW90] HAKEN, Hermann ; WUNDERLIN, Arne: Die Anwendung der Synergetik auf Musterbildung und Mustererkennung. In: KRATKY, Karl W. (Hrsg.) ; WALLNER, Friedrich (Hrsg.): *Grundprinzipien der Selbstorganisation*. Darmstadt : Wissenschaftliche Buchgesellschaft, 1990, S. 18–30

[KK92] KOHN, Wolfgang ; KÜPPERS, Günter: Die natürlichen Ursachen der Zwecke. In: *Selbstorganisation - Jahrbuch für Komplexität in der Natur-, Sozial- und Geisteswissenschaften* (1992), Nr. 3, S. 31–50

[Kra96] KRAMER, Ulrich: *Vom Elend des Fortschritts*. 1996. – online verfügbar unter http://www.vordenker.de/daselend/daselend.htm

[Mal01] MALSCH, Thomas: Zur Sozionik. In: *Sozionik Aktuell* (2001), Nr. 1. – online verfügbar unter http://www.sozionik-aktuell.de/

[Men98] MENGE, Hermann: *Langenscheidts Taschenwörterbuch Latein*. 48., Aufl. Berlin und München : Langenscheidts KG, 1998

[Mil02] MILBERG, Joachim: Erfolg in Netzwerken. In: MILBERG, Joachim (Hrsg.) ; SCHUH, Günther (Hrsg.): *Erfolg in Netzwerken*. Berlin : Springer, 2002, S. 3–16

[Mül00] MÜLLER, Johann-Adolf: *Systems Engineering*. Wien : Manz-Verlag Schulbuch (Fortis), 2000

[Pas92] PASCHE, Markus: Synergetik und Evolutorische Ökonomik / Universität Hannover Fachbereich Wirtschaftswissenschaften. 1992 ( 179). – Diskussionspapier

[Pit00] PITZ, Thomas: *Anwendung Genetischer Algorithmen auf Handlungsbäume in Multiagentensystemen zur Simulation sozialen Handelns*. Frankfurt am Main : Europäische Hochschulschriften, 2000

[PJS94] PEITGEN, Heinz-Otto ; JÜRGENS, Hartmut ; SAUPE, Dietmar: *Chaos: Bausteine der Ordnung*. Berlin : Klett-Cotta/Springer-Verlag, 1994

[Pop87] POPPER, Karl R.: *Das Elend des Historizismus*. 6., durchges. Aufl. Tübingen : J. C. B. Mohr (Paul Siebeck), 1987

[Rad04] RADEMACHER, Rochus: Organic-Computing wird Generaltechnik. In: *Computer Zeitung* (2004), Nr. 10, S. 1. – siehe auch weitere Artikel in dieser Ausgabe auf S. 6

[Sta94] STARK, Carsten: *Eine kritische Einführung in die Luhmannsche Systemtheorie*. Hamburg : Verlag Dr. Kovač, 1994

[Sta99] STAEHLE, Wolfgang H.: *Management: eine verhaltenswissenschaftliche Perspektive*. 8., ueberarb. Aufl. München : Franz Vahlens Verlag, 1999

[Stö94] STÖCKLER, Manfred: Selbstorganisation und Reduktionismus. In: *Selbstorganisation - Jahrbuch für Komplexität in der Natur-, Sozial- und Geisteswissenschaften* (1994), Nr. 5, S. 149–160

[Ver02] VERSTEEGEN, Gerhard (Hrsg.): *Software-Management: Beherrschung des Lifecycles*. Berlin : Springer, 2002

[War02] WARNECKE, Hans-Jürgen: Agilität im Wettbewerb erreichen - das Fraktale Unternehmen. In: MILBERG, Joachim (Hrsg.) ; SCHUH, Günther (Hrsg.): *Erfolg in Netzwerken*. Berlin : Springer, 2002, S. 263–274

[Wei01] WEIK, Elke ; LANG, Rainhart (Hrsg.): *Moderne Organisationstheorien*. Wiesbaden : Verlag Dr. Th. Gabler GmbH, 2001

[Wel99] WELLS, Don: *Homepage ExtremeProgramming.org*. 1999. – einige Grafiken in Anlehnung an Grafiken bei http://www.extremeprogramming.org/

[Wie71] WIENER, Norbert: *Kybernetik: Regelung und Nachrichtenübertragung in Lebewesen und Maschine*. 34. – 40. Aufl. Reinbek bei Hamburg : Rowohlt-Taschenbuch-Verlag, 1971

[WJR94] WOMACK, James P. ; JONES, Daniel T. ; ROOS, Daniel: *Die zweite Revolution in der Autoindustrie: Konsequenzen aus der weltweiten Studie aus dem Massachusetts Institute of Technology*. 8., durchges. Aufl. Frankfurt/Main; New York : Campus Verlag, 1994

[ZBGK01] ZUSER, Wolfgang ; BIFFL, Stefan ; GRECHENIG, Thomas ; KÖHLE, Monika: *Software Engineering: mit UML und dem Unified Process*. München : Pearson Studium, 2001

## Stichwortverzeichnis

## Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig durchgeführt und alle mir zuteil gewordenen Hilfen sowie das benutzte Schrifttum am Ende der Arbeit angegeben habe.

...................

Sebastian Stein