agile entwicklung und „emergence"

von jens coldewey

mehr zum thema:
www.agileprozesse.org

MANCHMAL IST MEHR DRIN, ALS MAN GLAUBT – AGILE ENTWICKLUNG UND „EMERGENCE"

Eines der Schlagwörter, die bei der Diskussion um agile Entwicklung immer wieder fallen, ist der Begriff der „Emergence". Dieser Begriff stammt nicht aus der Informatik, sondern aus der Komplexitätstheorie und lässt sich auf so unterschiedliche Systeme wie Evolution, Elektrodynamik, Spiele, Turingmaschinen und Nervensysteme anwenden. Zumindest ein grobes Verständnis dieses Phänomens erweist sich aber auch als hilfreich, wenn es gilt agile Verfahren zu verstehen.

der autor

Jens Coldewey (E-Mail: jens_coldewey@acm.org) ist unabhängiger Berater. Er hilft großen Organisationen bei der Einführung objektorientierter Konzepte und agiler Entwicklung.

Ich möchte Sie heute zu einem Spiel einladen. Keines von der Sorte „Senden Sie mir hundert Euro und überzeugen Sie ausreichend viele andere Trottel davon, Ihnen hundert Euro zu senden, dann werden Sie reich. Wenn Sie aussteigen, verdirbt Ihnen der Fluch von Atlantis die Kaffeemilch." Nein, ich möchte Sie zu einem kleinen Gedankenspiel einladen. Das Spiel ist sehr einfach, es besitzt nur vier Regeln:

1. Bausteine können sich reproduzieren.
2. Bei jedem Reproduktionsschritt unterscheiden sich die erzeugten Bausteine geringfügig und zufällig von dem erzeugenden.
3. Bausteine scheiden aufgrund statistischer Prozesse aus.
4. Die Unterschiede bei der Reproduktion beeinflussen die Wahrscheinlichkeit, mit der sich ein Baustein reproduzieren kann, und die Wahrscheinlichkeit, mit der er ausscheidet.

Für das Spiel ist es unbedeutend, wie die Bausteine aussehen und wie Reproduktions- und Ausscheidungsprozess von statten gehen. Um es aber spielen zu können, müssen wir konkreter werden. Ich möchte das Spiel daher zu einem Schachturnier ausbauen. Die Bausteine wären dann Algorithmen, die aus einer Stellung auf einem Schachfeld eine Zahl ermitteln, nicht zu verwechseln mit den Schachfiguren selbst. Solche Algorithmen werden in Schachprogrammen als Bewertungsfunktion verwendet, mit der aus der Fülle möglicher Züge der herausgesucht wird, der nach „Ansicht" des Programms die beste strategische Position bringt. Stellen Sie sich einen Algorithmus vor, der über eine große Anzahl von Parametern gesteuert wird, zum Beispiel, indem er im Sinne eines neuronalen Netzes jedes Feld abhängig von der Schachfigur, die auf ihm steht, gewichtet aufsummiert. Die jeweiligen Gewichtungsfaktoren bestimmen also den Algorithmus und beschreiben damit einen Baustein unseres Spiels.

Wie Sie sicher schon geahnt haben, bauen wir diesen Algorithmus nun in ein normales Schachprogramm ein[1]) und starten mit einer Reihe zufällig ausgewählter Bewertungsfaktoren, also mit einem zufällig ausgewählten Baustein. Das Schachprogramm wird vermutlich ausgesprochen schlecht spielen, etwa wie ein Spieler, der seine Figuren zwar entsprechend den Regeln aber sonst völlig zufällig über das Spielbrett bewegt. Aber der Schachalgorithmus stellt ja nur unsere Bausteine dar. Wir müssen jetzt also die vier Regeln noch umsetzen.

1. Algorithmen vermehren sich, indem eine Kopie von ihnen mit fast identischen Bewertungsfaktoren angelegt wird. In einem objektorientierten System ist das kein Problem.
2. Beim Kopieren wird auf einige, zufällig ausgewählte Faktoren ein normal verteilter Wert aufaddiert. Wir verändern also jeden Faktor um einen meistens kleinen Wert. Manchmal wird ein Faktor aber auch um einen großen Wert „korrigiert".
3. Zwei zufällig ausgewählte Algorithmen spielen gegeneinander Schach. Ein Algorithmus scheidet aus, wenn er mehr als eine bestimmte Anzahl von Niederlagen einstecken musste.
4. Algorithmen werden zufällig zur Reproduktion ausgewählt. Je mehr Siege sie bisher errungen haben, um so höher ist die Wahrscheinlichkeit, dass sie sich reproduzieren können; je mehr Niederlagen sie einstecken mussten, desto niedriger ist diese Wahrscheinlichkeit.

[1]) *Zusätzlich werden noch ein Algorithmus benötigt, der die erlaubten Züge kennt, und ein Verfahren, das diese Bewertungsfunktion mehrere Züge im Voraus anwendet und daraus dann den besten Zug ermittelt.*

Essenz

- Viele Systeme in der Natur und in den Wissenschaften zeigen Eigenschaften, die nicht direkt aus den ihnen zu Grunde liegenden Regeln ersichtlich sind.
- Dieses Phänomen wird als „Emergence" bezeichnet und ist ein grundsätzliches Verhalten von Systemen, in denen viele Agenten nach einfachen Regeln komplex interagieren.
- Viele agile Verfahren wurden mit möglichst wenigen Regeln als ein solches System konzipiert.
- Klassische Artefakte, wie zum Beispiel die Architektur, entstehen aus der Anwendung dieser Regeln und werden daher nicht mehr als eigene Arbeitsschritte erstellt.

Unser Spiel ist also nicht das einzelne Schachspiel, sondern eine Art Schachturnier zwischen verschiedenen Bewertungsfunktionen, die sich entsprechend den vier Regeln weiterentwickeln können oder vernichtet werden können.

Was passiert nun im Laufe der Zeit in unserem Turnier? Zunächst herrscht Chaos. Die Algorithmen spielen unsinniges Zeug zusammen und Siege sind eher zufällig. Nach einiger Zeit tauchen aber Algorithmen auf, die deutlich sinnvoller spielen. Sie gewinnen überdurchschnittlich oft gegen die „Chaoten" und verbessern dadurch ihre „Vermehrungschancen". Nach einiger Zeit beherrschen die sinnvolleren Algorithmen das Turnier. Es wird immer wieder neue Varianten geben, einige spielen schlechter als die anderen und werden nach wenigen Generationen verschwinden; andere nutzen geschickt Schwächen der anderen aus und werden sich durchsetzen. Wenn man lange genug wartet, werden eine oder wenige Bewertungsfunktionen entstehen, die exzellentes Schach spielen.

Dies ist kein reines Gedankenexperiment. Dieses Spiel wurde vor einiger Zeit auf über 5.000 Rechnern im Internet durchgeführt, wobei das Schachprogramm vier Züge voraussah. Es dauerte etwa vier Monate, bis sich Algorithmen entwickelt hatten, die nicht nur alle menschlichen Beteiligten an dem Projekt mühelos schlagen konnten, sondern auch alle anderen Schachprogramme, die vier Züge voraus sahen. Mit anderen Worten: Es war die beste Bewertungsfunktion für vier Spielzüge entstanden, die bis dato auf einem Rechner geschrieben wurden – praktisch von selbst. Zufall?

Um das zu ergründen, möchte ich Sie noch zu einem zweiten Spiel einladen, nach den gleichen Regeln, nur dieses Mal mit realen Bausteinen, statt virtueller Bewertungsfunktionen. Die Bausteine setzen wir diesmal aus Atomen der Elemente Wasserstoff, Sauerstoff, Kohlenstoff, Stickstoff und einigen anderen Elementen zusammen – wir arbeiten also mit organischen Molekülen. Einige erstaunlich einfache organische Moleküle haben eine spezielle Form, die andere Atome dazu veranlasst, sich dieser Form anzupassen, das Molekül also zu reproduzieren. Dafür benötigen wir eine Quelle für diese anderen Atome, also zum Beispiel eine Mischung aus Luft und Wasser, in der alle diese Elemente vorkommen. Damit die Reproduktion nicht identisch verläuft, bringen wir neben quantenmechanischen

$$\nabla \vec{E} = 4\pi\vec{\rho}$$

$$\nabla \times \vec{E} = -\frac{1}{c}\frac{\partial \vec{B}}{\partial t}$$

$$\nabla \vec{B} = 0$$

$$\nabla \times \vec{B} = \frac{4\pi}{c}\vec{j} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t}$$

Abb. 1: Die Maxwell'schen Gleichungen, aus denen sich die gesamte Elektrodynamik und die Optik herleiten lassen

Effekten noch UV-Licht und andere Strahlung von der Sonne und aus dem Weltraum in das Spiel, die hin und wieder ein Molekül zerschlägt oder manchmal auch nur geringfügig verändert. Damit sind alle vier Regeln erfüllt und wir können das System für eine Weile allein lassen, sagen wir für 1,5 bis 3 Milliarden Jahre. Das Ergebnis dieses Experiments könnte ähnlich aussehen wie das mit Abstand komplexeste System, das wir kennen: unser eigener Körper oder – um genauer zu sein – das Ökosystem auf der Erde mit allen voneinander abhängigen Organismen.

Emergent Behaviour

Es wird für Sie kaum Neuigkeitswert haben, dass die vier genannten Regeln das beschreiben, was wir Evolution nennen: ein einfacher Satz von Regeln, der aber zu den komplexesten Systemen führt, die wir kennen.

Es gibt noch einige weitere Beispiele für einfache Regeln, die zu sehr überraschendem Verhalten führen. So kennen die Physiker die „Maxwell'schen Gleichungen", vier sehr einfache Differentialgleichungen, die einige einfache Eigenschaften elektrischer und magnetischer Felder mathematisch beschreiben (**vgl. Abb. 1**). In ihnen steckt die Aussage, dass magnetische Nord- und Südpole niemals getrennt auftreten, dass alle Elektronen, die in einen stromdurchflossenen Raum hineinfließen, auch wieder hinausfließen müssen oder die Ladung des Raumelements erhöhen, und noch zwei etwas weniger intuitive Aussagen über den Zusammenhang zwischen magnetischen und elektrischen Feldern. Aus diesen wenigen und relativ einfachen Regeln (wenn man sich von den vierdimensionalen Differentialgleichungen nicht abschrecken lässt) lassen sich nun sämtliche Gesetze der Elektrodynamik ableiten: vom Zusammenhang zwischen Strom und Spannung über die Funktionsweise von Transformatoren und Schwingkreisen bis hin zu den Brechungsgesetzen des Lichts. Auch hier zeigen einige in ihrer Grundaussage schon fast triviale Regeln ungeahnte Komplexität.

In der Komplexitätsforschung wird solch überraschend auftauchendes Verhalten als *Emergent Behaviour* bezeichnet (vgl. [Hol98]). Dieses *Emergent Behaviour* ist eine grundlegende Eigenschaft von Systemen, in denen viele Agenten nach einfachen Regeln miteinander interagieren. Weitere Beispiele sind das Verhalten der Börse, staatenbildende Insekten oder in unserer Branche die Leistungsfähigkeit von Computern gemessen an ihrer relativ einfachen Grundstruktur: Die einfachsten Computer, sogenannte Turingmaschinen, bestehen aus einem Band, auf das ein Schreib- und Lesekopf Zeichen schreiben kann, und aus einem Programm, mit dem der Kopf auf dem Band vorwärts und rückwärts positioniert werden kann sowie Zeichen gelesen und geschrieben werden können. Es lässt sich nachweisen, dass diese einfache Maschine alle Probleme lösen kann, die sich überhaupt mechanisch lösen lassen, vorausgesetzt man gibt ihr genügend Zeit. „Kleinere technische Optimierungen", wie moderne Prozessoren dienen „nur" dazu, die Bearbeitungszeiten von wenigen Jahrtausenden auf Millisekunden zu senken.

Schwieriger, als ein komplexes System mit *Emergence* zu entdecken, ist es allerdings, ein solches System zu konzipieren. John Holland beschreibt in seinem Buch „Emergence", wie Maxwell seine legendären Gleichungen gefunden hat (vgl. [Hol98]): Er hat versucht, sich die Elektrodynamik als Flüssigkeit vorzustellen und dann nach übertragbaren Eigenschaften gesucht. Maxwell hat also

[2])Dass Maxwell damals von der Idee des Äthers geleitet wurde, in dem sich elektromagnetische Wellen ausbreiten, tut der Gültigkeit seiner Ergebnisse im nicht-relativistischen Bereich keinen Abbruch. Auch wenn der Äther sich später als falsches Modell herausgestellt hat, führte die Metapher Maxwell doch in die richtige Richtung.

In eigener Sache

Was ich bereits mit dem Start dieser Kolumne vor einem Jahr angedeutet hatte, haben wir jetzt umgesetzt: Ab dieser Folge läuft die Kolumne unter dem Titel „Agile Entwicklung”. Wir folgen damit der Erkenntnis, dass „agile Entwicklung” der wesentlich treffendere Begriff ist, der sich im letzten Jahr auch allgemein durchsetzen konnte. Darüber hinaus werden Sie diese Kolumne aber weiterhin im gewohnten Gewand vorfinden.

eine *Metapher* verwendet, die ihm die Möglichkeit gab, ein bereits bekanntes System mit *Emergence* auf das noch unbekannte System der Elektrodynamik anzuwenden[2]). Indem er die ihm zur Verfügung stehenden *Bausteine* der Mathematik dann geeignet *kombinierte*, erhielt er schließlich sein System von Regeln. Das eigentlich Potenzial dieses System wurde aber erst im Laufe der Zeit erkannt, als sich aus ihm immer mehr bekannte Regeln ableiten ließen.

Agile Entwicklung

Hier kommt nun endliche die agile Softwareentwicklung ins Spiel. Beim Entwurf der meisten dieser Verfahren standen die Überlegungen der Komplexitätstheorie Pate: Die „Komplexen Adaptiven Systeme” waren Ideengeber des „Adaptive Software Developments” (vgl. [Hig00]). Kooperative Spiele sind die Metapher der „Crystal”-Methodenfamilie (vgl. [Coc02]). Ken Schwaber verwendete die chemische Verfahrenstechnik als Metapher, um „Scrum” zu entwickeln (vgl. [Sch02]). Kent Beck ging für „Extreme Programming” von alltäglichen Regelungsprozessen aus, wie dem Autofahren (vgl. [Bec00]). Extreme Programming empfiehlt die gleiche Technik übrigens auch für das Projekt selbst: Die Metapher bildet eine der Prinzipien des Verfahrens. Doch dazu gleich mehr.

Auch die traditionellen Prozesse des Software-Engineering nutzen Metaphern, insbesondere die mittlerweile arg überstrapazierte Metapher des Gebäudebaus. Dass diese Metapher bei der heutigen Softwaretechnik mehr verschleiert als sie aufdeckt, hatte ich früher bereits diskutiert (vgl. [Col00]). Es sind also eher die neuen Metaphern wie adaptive Systeme, kooperative Spiele oder Regelungsprozesse, die agile Verfahren von traditionellen unterscheiden, nicht so sehr ihr grundsätzlicher Einsatz.

Ganz anders ist aber das weitere Vorgehen bei agilen Verfahren. Statt detailliert Artefakte, Aktivitäten und Rollen vorzugeben, die auf manchmal über tausend Seiten beschrieben werden, beschränken sich agile Verfahren auf wenige Regeln oder Prinzipien, aus denen sich der Projektverlauf dann von selbst ergibt, also als *Emergent Behaviour*. Wer den Einsatz traditioneller Prozesse in der Praxis verfolgt, findet oft genug Diskussionen, ob jetzt der Einsatz eines bestimmten Formulars oder Artefakts notwendig sei oder nicht. Diese Projekte entsprechen damit Hollands Beobachtung, dass „punktuelles Interesse den Blick für die globalen Zusammenhänge verstellt” (vgl. [Hol98], Seite 212). Agile Verfahren arbeiten gegen diese Versuchung, indem sie die Details gar nicht erst zur Verfügung stellen. Das Verfahren gibt Werte und Prinzipien vor, keine detaillierten Arbeitsanweisungen.

Architektur und Emergence

Schön kann man das am Beispiel der Architektur sehen. Motiviert von dem hinkenden Vergleich mit der Bauindustrie messen traditionelle Verfahren der explizit zu Projektbeginn schriftlich fixierten Architektur eine sehr große Bedeutung bei. So schrieb Barry Boehm 1996: „Ich kann nicht genügend betonen, wie kritisch der ‘Life Cycle Architecture’ Meilenstein für Ihr Projekt und Ihre Karriere ist. Wenn Sie die Kriterien des LCA Meilensteins nicht erfüllt haben, gehen Sie auf keinen Fall in die volle Entwicklung. Versammeln Sie die ausschlaggebenden Entscheidungsträger und erarbeiten Sie einen neuen Projektplan, der die LCA Kriterien erfüllt” (vgl. [Boe96], zitiert nach [Jac99]). Als mir Alistair Cockburn vor fünf Jahren das erste Mal die Grundzüge agiler Entwicklung erklärte, war meine erste Frage: „Was ist mit der Architektur? Man kann ohne Architektur doch keine professionelle Softwareentwicklung betreiben!” „Probiere es aus, es funktioniert” war Cockburns Antwort, die man etwas wissenschaftlicher so formulieren könnte: Die Architektur entsteht in einem agilen Projekt nicht als eigenes Artefakt, sondern als *Emergent Behaviour* aus den Regeln des Verfahrens und der Kollaboration des Teams.

Die Praxis zeigt, dass dies für alle agilen Verfahren gilt. Besonders gut kann man diese Aussage beim *Extreme Programming (XP)* nachvollziehen (vgl. [Col02]). Ausgehend von den Grundwerten „Streben nach Einfachheit” und „Qualitätsarbeit leisten” sind mindestens fünf der dreizehn Praktiken verantwortlich für das Entstehen einer guten Architektur:

- Metapher
- Einfaches Design
- Programmieren in Paaren
- Entwicklertests
- Refactoring

Eine Metapher wie die bekannte Desktop-Metapher „der Computer ist wie ein Schreibtisch” ist die Grundlinie der gesamten Entwicklung. Über die grundsätzliche Bedeutung von Metaphern in kreativen Prozessen schreibt Holland: „Selbst eine einfache Metapher [...] umfasst viel mehr, als die Quelle [hier z. B. der Schreibtisch] und das Ziel [hier das System] [...] Beide [...] sind umgeben von Bedeutung und Assoziationen. Die Metapher bewirkt eine Art Rekombination dieser Umgebungen und verbessert die Wahrnehmung sowohl der Quelle als auch des Ziels. Wenn die Metapher erfolgreich ist, ergeben sich aus ihrer Interaktion Verbindungen und Interpretationen, die vielfältig, überraschend, amüsant oder begeisternd sein können.” (vgl. [Hol98], Seite 208). So revolutionierte die an den Xerox-Laboratorien erdachte und von Apple zur Marktreife gebrachte Metapher vom Desktop die Computerwelt und war die Voraussetzung für den Computer als Massenprodukt – auch wenn letztlich schlechte Umsetzungen dieser Metapher den Markt übernommen haben. Eine solche Metapher ist also ein Herzstück eines XP-Projekts und Voraussetzung für das Entstehen einer Architektur.

Die weiteren vier Praktiken bilden den inneren Zyklus eines XP-Projekts und dienen nicht nur dem Erzeugen von Code sondern zugleich dem Aufbau der Architektur. Einfaches Design erlaubt es, den Überblick zu behalten. Programmieren in Paaren und Entwicklertests bilden nicht nur ein engmaschiges Netz für die Qualitätssicherung, sondern berücksichtigen zugleich, dass zwei Personen gemeinsam eine höhere Kreativität zeigen als jeder für sich. So wird der Design- ▸

prozess verbessert. Zudem bilden diese beiden Praktiken die Voraussetzung, dass Refactoring funktioniert, also die ständige inkrementelle Verbesserung der Architektur. Die Eleganz der durch diese Kombination entstehenden Architekturen erstaunt häufig selbst das an ihm arbeitende Team: ein klassisches Beispiel von *Emergence*.

Aber ist es nicht sehr unwahrscheinlich, dass man so eine optimale Architektur findet? Sicher, aber „diese Unwahrscheinlichkeit ist keine all zu starke Einschränkung, wenn wir *nicht* nach Perfektion suchen" schreibt Holland (vgl. [Hol98]). Tatsächlich streben agile Verfahren, keine perfekte Software an, sondern haben als Ziel lediglich, Software wirtschaftlich zu erstellen. Da die Software gut änderbar ist, kann man Probleme immer noch beheben, wenn sie auftreten. Nebenbei zeigt die Erfahrung, dass die bei agiler Entwicklung entstehenden Architekturen den Reißbrettentwürfen traditioneller Prozesse oft deutlich überlegen sind, da sie meist schlanker, übersichtlicher und leichter zu ändern sind.

Noch eine weitere Beobachtung Hollands ist für agile Entwicklung von Bedeutung: „Erst wenn Sie mit den einzelnen Bausteinen Ihrer Branche so vertraut sind, dass Sie nicht mehr darüber nachdenken müssen, wie sie zusammengesetzt werden, erreichen Sie die kreative Phase" (vgl. [Hol98], Seite 212). Das trifft eine der Kritiken, die immer wieder an agiler Entwicklung geübt wird: Die Verfahren setzten ein gutes Team voraus. Niemand käme auf die Idee, eine Symphonie einzuspielen und die Musiker dafür in einem Kurs „Trompete in sieben Tagen" auszubilden. Nur die Softwareentwicklung scheint dafür geeignet zu sein, mit Laien professionell arbeiten zu können, zumindest wenn man dem Eindruck folgt, den so mancher Verfechter traditioneller Prozesse zu erwecken versucht.

Zum Schluss

Ein kleiner Nachtrag bleibt mir noch zu einem Buch, auf das ich in meiner Kolumne in OBJEKTspektrum 6/2001 ([Col01]) hingewiesen hatte. Damals hatte ich die vierte Ausgabe von Bernd Oestereichs Buch „Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language" empfohlen. Bernd Oestereich hat mich freundlicherweise mit Hilfe eines Belegexemplars als Bestechungsgeld darauf hingewiesen, dass mittlerweile die fünfte Auflage dieses Buchs die aktuelle Fassung ist.

Natürlich freue ich mich auch diesmal über Ihre Ideen und Anregungen, die Sie per Leserbrief an die Redaktion oder per E-Mail direkt an mich senden können. ■

Literatur

[Bec00] K. Beck, Extreme Programming Explained – Embrace Changes, Addison-Wesley 2000

[Boe96] B.W. Boehm, Anchoring the Software Process, IEEE Software, Juli 1996

[Coc02] A. Cockburn, Agile Software Development, Addison-Wesley 2000

[Col00] J. Coldewey, Über Sieben Brücken musst Du geh'n, in: OBJEKTspektrum 06/2000

[Col01] J. Coldewey, Beständig ist nur der Wandel – Agile Entwicklung und Änderbarkeit, in: OBJEKTspektrum 6/2001

[Col02] J. Coldewey, Multi-Kulti: Ein Überblick über die agile Entwicklung, in: OBJEKTspektrum 01/2002

[Hig00] J.A. Highsmith III, Adaptive Software Development – A Collaborative Approach to Managing Complex Systems, Dorset House 2000

[Hol98] J.H. Holland, Emergence – From Chaos to Order, Oxford University Press, 1998

[Jac99] I. Jacobson, G. Booch, J. Rumbaugh, The Rational Unified Software Development Process, Addison-Wesley 1999

[Sch02] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall 2002