Agile Software Entwicklung nach Winston Royce

Jens Himmelreich
neuland – Büro für Informatik

Abstract

Es wird nach den Grundlagen des sogenannten Wasserfallmodells gefragt. Seine Quelle, Winston W. Royce Arbeit Managing the Development of Large Software Systems*, wird referiert. Das Verhältnis dieses Ansatzes der Softwareentwicklung zu agilen Ansätzen wird untersucht. Dabei wird die These vertreten Royce folge durchaus agilen Werte, tue dies aber mit den gedanklichen Mitteln seiner Zeit.*

Das Wasserfallmodell ist ein verbreiteter Ansatz in der Softwareentwicklung. Agile Softwareentwicklung hat sich zunächst gegen dieses Modell etabliert. Was ist dieses Wasserfallmodell, wer formulierte es und worin besteht sein Gegensatz zur agilen Softwareentwicklung? Ist dieser Gegensatz absolut oder besteht die Möglichkeit eine Brücke zu schlagen? Diesen Fragen will dieses Papier nachgehen.

1. Das Wasserfallmodell

Die deutschsprachige Wikipedia fasst den Ansatz des Wasserfallmodells wie folgt zusammen: "Das Wasserfallmodell bezeichnet ein Vorgehensmodell in der Softwareentwicklung, bei dem der Softwareentwicklungsprozess in Phasen organisiert wird. Dabei gehen die Phasenergebnisse wie bei einem Wasserfall immer als bindende Vorgaben für die nächste tiefere Phase ein." [Wik 06] Das Wasserfallmodell ist also ein Vorgehensmodell für die Softwareentwicklung. Die Phasen sind "wie bei einem Wasserfall" angeordnet. Die Anordnung der Phasen wird durch das Bild bestimmt und anschaulich gemacht.

Die Wikipedia fährt fort: "Zuerst vorgeschlagen wurde es [das Wasserfallmodell] in einem Papier aus dem Jahre 1970 von Winston Royce mit dem Titel 'Managing the Development of Large Software Systems: Concepts and Techniques'." (Ebenda) Dieser Aufsatz, die Grundlage des Phasenmodells und die Quelle des Wasserfalls, wird uns im Weiteren beschäftigen. Wie der Wasserfall entspringt und wie der Softwareentwicklungsansatz gründet, werden wir untersuchen.

"Im Wasserfallmodell hat jede Phase wohldefinierte Start- und Endpunkte mit eindeutig definierten Ergebnissen. In Meilensteinsitzungen am jeweiligen Phasenende werden die Ergebnisdokumente verabschiedet. Zu den wichtigsten Dokumenten zählen dabei das Lastenheft sowie das Pflichtenheft. In der betrieblichen Praxis gibt es viele Varianten des reinen Modells. Es ist aber das traditionell am weitesten verbreitetste Vorgehensmodell." (Ebenda)

Das Wasserfallmodell ist damit skizziert. Verschiedene Phasen beschreiben den Softwareentwicklungsprozess. Die Phasen folgen einander. Eine Phase muss definiert abgeschlossen sein, damit mit der nächsten begonnen werden kann. Das Bild einer Kaskade ergibt sich. Die Namen der Elemente der Kaskade variieren, bezeichnen jedoch ungefähr immer das gleiche: Anforderung, Spezifikation, Design, Programmierung, Testen, Auslieferung und Betrieb.

Der Gegensatz zu agilen Ansätzen ist implizit benannt. Am Beispiel des agilen Manifests als Grundlage unterschiedlicher Weisen, agile Softwareentwicklung zu betreiben, zeigen sich Differenzen. Das Wasserfallmodell definiert sich über den Prozess, noch enger: es definiert sich über die Mechanik der Phasenabfolge. Ohne diese Phasenabfolge verliert es seine Identität. Ganz im Gegensatz dazu der erste agile Wert. "Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge" [Coc 03, 281] Der zweite Wert "Funktionierende Software vor ausführlichen Dokumentationen" widerspricht dem für das Wasserfallmodell essentiellen Verständnis der Dokumentation als Abschluss einer Phase bzw. als Bewegungsform des Wissens in einem Projekt. Da die Dokumente einer Phase die verbindliche Grundlage der nächsten Phase sind, steht das Wasserfallmodell auch im Gegensatz zum vierten agilen Wert: "Reagieren auf Änderungen ist wichtiger als das Befolgen eines Plans." (Ebenda) Der Kunde kommt im Wasserfallmodell nicht als permanenter Akteur vor. Er ist an der Erarbeitung der Anforderungen beteiligt und wird dann erst wieder in der Phase der Auslieferung hinzugezogen. Kundenbeteiligung, wie sie im dritten agilen Wert ("Kundenbeteiligung ist wichtiger als Vertragsverhandlungen") benannt ist, widerspricht der inneren Logik der Phasen. Da die erste Phase, Anforderungsdefinition, abgeschlossen ist, spielt der Kunde keine Rolle im Entwicklungsprozess. Es wäre ein Symptom für einen Defekt des Prozesses, wenn in der Phase der Programmierung auf Kunden zurückgegriffen werden müsste.

Dieser Gedanke - Kundenbeteiligung als Symptom eines scheiternden Entwicklungsprozesses - lässt sich noch verallgemeinern. Der Zusammenhang, den die agilen Werte zwischen zwei Elementen herstellen, und bei dem sie die rechte Seite, die Seite der Zusammenarbeit und der Interaktion, betonen, müsste aus der Sicht des Wasserfallmodells umgedreht werden. Aus dem Befolgen von Prozessrichtlinien bekommen erst die Individuen und Interaktionen ihre Rolle und ihre Funktion im Prozess. Erst aus der Betonung von ausführlichen Dokumentationen entsteht die Möglichkeit funktionierender Software. Erst saubere, vertraglich regulierte Schnittstellen erlauben ein explizites Verständnis der Kundenwünsche. Und nur das Befolgen eines Plans schafft die Möglichkeit, den Kunden zufrieden zu stellen.[1]

[1] Der Gegensatz der Ansätze ist deutlich. Es könnte ein interessantes Unterfangen darstellen, ihn unter Bezug auf die Kategorien des Impliziten und des Expliziten zu rekonstruieren. Das Wasserfallmodell scheint um den Begriff des expliziten Wissens zu kreisen, für sein Selbstverständnis scheint die Explikation und Verdinglichung von Wissen substantiell zu sein. Der agile Ansatz bringt den impliziten Charakter des Wissens an entscheidender Stelle ins Spiel. Wenn Wissen sich nicht vollständig oder noch nicht einmal ausreichend tragfähig explizieren und in geronnene Form, in der Regel eine Dokumentation, verwandeln ließe, wäre es wichtig, die Träger und Agenten dieses Wissens ins Spiel zu bringen. Das implizite Wissen würde durch seine Träger in den Prozess integriert.

Das Wasserfallmodell ist das "traditionell am weitesten verbreitetste Vorgehensmodell" [Wik 06]. Der Gegensatz zur agilen Softwareentwicklung ist deshalb wichtig, weil sich Agilität immer gegen das Wasserfallmodell durchsetzen muss.[2] Die Fragen, was im Allgemeinen unter dem Modell verstanden wird, wie es formulierte wurde und was eigentlich seine Intention war, werden uns im Folgenden begleiten.

2. Das Wasserfallmodell als visuelles Paradigma

Ausgehend von der Feststellung, dass der Begriff des Wasserfallmodells im Schlüsseltext von Winston Royce nicht einmal benutzt wurde und ausgehend von der weiter unten zu belegenden Behauptung, dass das heutige Verständnis des Wasserfallmodells (siehe [Wik 06]) so nicht bei Royce formuliert wurde, stellt sich die Frage nach der Kraft des Begriffs.

Die eine Hälfte der Frage ist die nach der Urheberschaft. Wer eigentlich hat wann den Begriff des Wasserfallmodells geprägt? Diese Frage kann hier nicht beantwortet werden. Aber eine Beobachtung aus der Lektüre soll beigetragen werden. Wer den Aufsatz 'liest' versteht sofort warum vom Wasserfallmodell gesprochen wird. Es ist die Kraft der Metapher, die für die Evidenz der Bezeichnung bürgt. Wasser ist die Standardmetapher für einen Prozess, für etwas das im Fluss ist. Und wenn dieser Prozess Phasen durchläuft, zu denen er nicht zurückkehren kann, wenn ein Niveau von einem anderen abgelöst wird und es kein Zurück gibt, dann symbolisiert der Wasserfall am Besten den Fluss. Der Wasserfall verbindet die Niveaus auf die Art des Wassers: Wasser kann nur in eine Richtung fließen, es kann nur Fallen, von einem Niveau auf ein anderes. Wenn demnach Phasen mit dieser irreversiblen Flussrichtung angeordnet werden, drückt der Wasserfall das am besten aus.

Diese Logik drück sich auf eine eigentümliche Weise in Winston Royce Aufsatz aus. Den Aufsatz kann man nicht im engeren Sinne des Wortes lesen. Man kann ihn nur ansehen und lesen. Mehr als die Hälfte der Textfläche - welch ein komisches Maß für das Lesen - ist von Grafiken bedeckt. Royce will den Leser nicht nur überreden, er will ihn mit visueller Evidenz für seine Thesen gewinnen. Insbesondere die erste Visualisierung des gesamten Prozesses drückt das aus, was im Allgemeinen unter einem Wasserfall verstanden wird. Es handelt sich dabei um ein Treppenmuster von Phasen, die so angeordnet sind, dass nur noch das Wasser eingefüllt werden müsste, um den Wasserfall vollständig zu machen. Pfeile verbinden die Phasen in der Richtung, die ein von der Schwerkraft angezogener Wasserstrom nehmen würde.

Ob Royce bei seinen Bildern an einen Wasserfall dachte sei dahingestellt. Fest steht, dass er seine Phasen so anordnet, dass sie einem Wasserfall gleichen. Die Abfolge gewinnt damit eine Evidenz, die rein textlich nicht herzustellen wäre. Der Begriff Wasserfallmodell bringt das nur auf den Punkt. In ihm verdichtet sich die Argumentation der Notwendigkeit von Phasen, von strikt linearisierten Phasen mit der

[2] Umso mehr muss man es den Autoren des agilen Manifestes danken, dass sie den Prozess der agilen Softwareentwicklung positiv definiert haben.

sichtbaren Anordnung dieser Phasen von links nach rechts und von oben nach unten. Der Wasserfall ist eine Metapher, die diesen Wesenszug des Modells auf den Punkt bringt.

Eine Metapher an derart strategischer Position birgt auch eine Gefahr. So komprimiert wie sie ein Modell kommuniziert, so kondensiert wie sie es ermöglicht den Sachverhalt zu erinnern, so verkürzt kann sie auch sein. Die Metapher kann ein derartiges Eigenleben entwickeln, dass es dem Urbild nicht mehr möglich ist, sich aus den Fesseln ihrer Plausibilität zu befreien. So wirkt es auch am Beispiel von Royce Text. Obwohl Royce um Differenzierungen ringt, obwohl er den Wasserfall nur zum Ausgangspunkt einer Betrachtung macht, die er im Folgenden relativieren und durchstreichen wird, wird ihm nur eine Vaterschaft zuerkannt - und das ist die Vaterschaft am Wasserfallmodell.[3]

Wenn denn das Wasserfallmodell ein visuelles Kondensat darstellt, was ist dann das agile Gegenstück? Was ist das visuelle Pendant der agilen Softwareentwicklung? Es ist der Kreis. Der Wasserfall, die Kaskade, die Linearität gegen den Kreis. Kreis und Iteration als Sinnbilder des agilen Prozesses? Das scheint zu passen

3. Das Wasserfallmodell nach Winston Royce

Larman und Basili sprechen in ihrer Rekonstruktion der Geschichte der iterativen und inkrementellen Softwareentwicklung davon, dass viele Menschen Royce Papier fälschlicherweise als Ausdruck des 'single-pass' Wasserfallmodells begreifen würden.[4] In Wirklichkeit entwickle er einen Ansatz der davon unterschieden sei (vgl. [Lar 03, 48]). Royce selber - und damit sind wir endlich beim eigentlichen Papier angekommen -

---

[3] Es stellt sich die Frage nach der Kraft dieser Metapher. Warum ist es genau dieses Bild, das derart starr die Interpretation eines Modells beherrscht, das - sich selbst zur Folge - anders sein will. Die Kraft des Bildes, die Kraft des Symbols muss eine Beziehung zu etwas anderem unterhalten, sonst wäre ihre Macht nicht zu erklären. Die Frage nach diesem anderen kann hier nicht beantwortet werden, es soll lediglich eine These formuliert werden, die in die Richtung einer Antwort weist. Die Antwort zu finden - sofern das in diesem Feld überhaupt möglich ist - bedarf eines viel größeren Aufwandes und vermutlich auch eines anderen Apparates, der hier nicht zur Verfügung steht.
Um hier eine Vermutung zu äußern, ist es vielleicht hilfreich, die Frage noch einmal zu formulieren. Was verleiht dem Bild des Wasserfallmodells seine Stärke, was macht das so (s.o.) verstandene Modell so verbreitet. Ein Aspekt der Antwort ist schon formuliert. Es ist die große Prägnanz dieser Metapher, die den Sachverhalt derart konzentriert ausdrückt und derart kondensiert erinnerbar macht. Auf der Ebene des Gegenstandes - so die eigentliche Vermutung - ist es Parallelität, Kongruenz (zwei hoffentlich unscharfe Begriffe, die mehr Fragen stellen als sie beantworten, wenn es um die Natur dieser Beziehung geht) von Bild/Metapher/Modell und Herstellungskultur/Paradigma der Gesellschaft. Es ist die - äußerst grobschlächtig formuliert – Weise des ingenieurmäßigen Planens und Herstellens, die in unserer Industriekultur herrscht (hier der emphatische Ingenieurbegriff des Phasenmodells der 50er). Der Begriff der Industriekultur soll in eine Richtung deuten, die andernorts unter dem Begriff Fordismus zum Thema gemacht worden ist und wiederum eine Verwandtschaftsbeziehung von Kritik des Fordismus und Agilität vermuten lässt. Diese Verwandtschaftsbeziehung hört auf den Namen Lean und ist in der Debatte um agile Softwareentwicklung gar nicht zufällig unter dem Label Lean Programmming unterwegs [Pop 03].

[4] “Many – incorrectly – view Royce's paper as the paragon of single-pass waterfall.” [Lar 03, 48]

spricht davon, dass seiner Erfahrung nach der einfachere Ansatz - single-pass Wasserfall[5] - bei größeren Softwareprojekten nie funktioniert habe: "In my experience, however the simpler method never worked on large software development efforts [...]." [Roy 70, 335]

Wenn es so ist, wenn Royce der erste Kronzeuge gegen das klassische Wasserfallmodell ist, ist zu fragen, welches Modell er denn ausführe, welches Modell denn entwickelt werde, in gerade dem Aufsatz, der doch - so hieß es - die Quelle des Wasserfalls und seines Modells sei.

Das erste Bild seines Aufsatzes, das den kompletten Prozess darstellt, beschreibt genau jenen Wasserfall, der in die Literatur und in die Köpfe eingegangen ist. Die Phasen heißen: System Requirements, Software Requirements, Analysis, Program Design, Coding, Testing, Operations. Die einzelnen Phasen werden mit einfachen Pfeilen verbunden. Die Phasen sind in einem abfallenden Treppenmuster dargestellt, gerade so, wie es ein Wasserfall verlangt.

Royce fährt sofort mit einem anderen Bild fort. Es sieht aus wie das letzte, nur dass die Phasen jetzt mit Pfeilen verbunden sind, die bidirektional sind. Er spricht von einem "iterative relationship" (ebenda, S. 328) zwischen den Phasen. Was den Phasen jetzt nur noch ihre Ordnung gibt, ist ihre Anordnung in der Treppenstruktur. Royce beschreibt den Sinn dieser Verfeinerung damit, dass er ausdrücken wolle, in jedem Prozessschritt gäbe es eine Wechselwirkung mit dem letzten und dem nächsten Schritt, aber selten mit einem anderen. (Ebenda) Der Vorteil dieser Beschränkung der Wechselwirkung sei, dass Änderungen auf handhabbare Bereiche beschränkt werden könnten. Es gäbe so etwas wie eine bewegliche Grundlinie, die einen möglichst großen Teil der bisherigen Arbeit rette, wenn es zu Änderungen komme. (Ebenda)

Royce glaube zwar an dieses Konzept, aber die bisher beschriebene Form sei riskant und fehleranfällig. Erst in der Testphase würden zum ersten Mal die Ergebnisse der Analysephase überprüft. Die Phänomene, die zu analysieren gewesen wären, seien nicht präzise analysierbar. (Das ist bei Royce ein kategorisches, kein empirisches Argument.) Es gäbe keine Lösungen wie etwa in der Mathematik. Wenn also die getroffenen Annahmen an äußeren Bedingungen scheiterten[6], dann sei entweder das Design zu überarbeiten oder sogar zu den Anforderungen zurückzukehren, so dass diese neu zu fassen wären. Als Resultat Kehre der Prozess zu seinem Ursprung zurück und es sei zu vermuten, dass das Projekt seine Erwartungen in Kosten und Dauer um 100% überschreite.

Erst jetzt hat Royce das Problem entwickelt, zu dem er Stellung nehmen möchte. Seine Vorrede ist quasi abgeschlossen. Das was er bisher entwickelt hat ist ein Modell, das er mit dieser Arbeit nicht erfinden will, welches er nicht einmal darstellen möchte, sondern an dem er eine Korrektur, eine Ergänzung anbringt. Er verhält sich nicht zu den Ursprüngen dieses Modells. Es ist einfach da. Vielleicht wird man ihm am gerechtesten,

---

[5] Mit was für einem Bild haben wir es eigentlich hier zu tun? Das unverfängliche Attribut single-pass führt eine Unterscheidung ein, die für einen Wasserfall sinnlos ist.

[6] Es sei angemerkt, dass hier sehr stark die physikalische Dimension des Rechnens im Blick ist - Raum und Zeit des Computing.

wenn man das Modell einem *Common Sense* der damaligen Softwareentwicklung zuschreibt, einem Diskurs, in dem jede Auseinandersetzung über Softwareentwicklung situiert war, die damals stattfand. Das Wasserfallmodell wäre dann so etwas wie ein Horizont, vor dem sich alle Debatten abspielen, ein Horizont, der den stattfindenden Debatten erst ihren Sinn gibt.

Was aber, so muss nun gefragt werden, ist der Sinn von Royce Aufsatz? Royce fährt gleichsam programmatisch und kategorisch fort: "However, I believe the illustrated approach to be fundamentally sound. The remainder of this discussion presents five additional features that must be added to this basic approach to eliminate most of the development risks." (Ebenda, S. 329) Royce distanziert sich nicht vom beschriebenen Wasserfallmodell, aber er zeigt fünf Punkte auf, die ergänzt werden müssen, damit die meisten der Entwicklungsrisiken eliminiert werden könnten.

4. Royce Ergänzungen zum klassischen Wasserfallmodell

Step 1: Program Design comes first

Zwischen die Anforderungs- und die Analysephase solle eine vorbereitende Designphase eingefügt werden. Erfahrene Programmdesigner sollten ein erstes Design entwerfen, um möglichst früh zu wissen, ob die Anforderungen nicht schon an einfachen Bedingungen des System scheitern. Sei dies der Fall, müsse zur Anforderungsphase zurückgekehrt werden. Ergebnis dieser vorläufigen Designphase sollte ein Überblicksdokument sein, das für den weiteren Prozess ein elementares Verständnis des gesamten Systems entwickle.

Step 2: Document the Design

Royce hat eine eindeutige Haltung zum Dokumentieren: "quite a lot" (ebenda, S. 332). Für Royce ist eine ausführliche Dokumentation die Schlüsselfrage des Gelingens. Sobald die Dokumentation nicht mehr stimme, sei alles zu stoppen und die Dokumentation auf den Stand zu bringen. Warum? Royce führt mehrere Gründe an. Erstens sei der Designer zu einer eindeutigen Position zu nötigen, gerade weil er mit den verschiedenen, beteiligten Parteien verhandle. Mündliche Absprachen seien zu ungenau und zu zerbrechlich. Zweitens ist in der frühen Phase die Dokumentation auch die Spezifikation und das Design. Bis die Programmierung beginne, meinen die drei Begriffe das gleiche Ding. Drittens zeige sich der monetäre Gewinn einer guten Dokumentation in den Phasen Test, Betrieb und Redesign. In der Testphase kann man sich mit guter Dokumentation auf die Fehler konzentrieren, man muss nicht das Verständnis des Systems aus der Anwendung ableiten. In der Betriebsphase können einfachere Arbeitskräfte das Programm bedienen. Ohne Dokumentation müssten das diejenigen machen, die es entwickelt hätten. Mit einer guten Dokumentation kann das Redesign effektiv betrieben werden. Ohne eine solche muss oftmals das gesamte

Programm reimplementiert werden. Royce stellt sechs Dokumentationen vor, die im Prozess zu entwickeln seien.

Step 3: Do it twice

Wenn es sich bei dem Programm um ein neues handle, dann sei der zweit wichtigste Punkte nach der Dokumentation, dass das an den Kunden ausgelieferte Programm das Zweite sei. Nach der Phase des vorläufigen Designs, sei das Programm einmal prototypisch zu implementieren. Das müsse fehlerfrei geschehen, brauche aber nicht akademischen Ansprüchen genügen. Die kritischen Stellen seien auszuprobieren, Hypothesen zu testen, so dass ein intuitives Gefühl für das echte Programm entstehe. (Ebenda, S. 334) Royce veranschlagt einen relativ langen Zeitraum für diesen Prototyp. In seinen Beispielen sind es ein Viertel bis ein Drittel der Gesamtlaufzeit des Projekts, die mit dem Prototyp zugebracht werden.

Step 4: Plan, Control and Monitor Testing

Ohne Frage sei die Testphase der größte Nutzer der Projektressourcen. Royce macht folgende Vorschläge für die Testphase: Erstens sollen Testspezialisten testen, nicht die Designer. Das funktioniere, wenn die Dokumentation entsprechend sei. Zweitens fielen viele Fehler im Code schon bei einem einfachen Gegenlesen auf, welches jemand durchführen müsse, der nicht am ursprünglichen Code beteiligt sei. Drittens müssten alle Pfade des Programms mit definiertem Input und Output getestet werden. Und viertens müsse dann das Management den richtigen Zeitpunkt und die richtige Art und Weise finden, um das finale Auschecken durchzuführen.

Step 5: Involve the Customer

Es sei wichtig den Kunden auf formelle Art am Prozess zu beteiligen. Das Design habe immer Spielraum für Interpretationen. Wenn der Kunde zwischen Anforderungsdefinition und Betrieb nicht einbezogen würde, gäbe es Ärger. Der Kunde müsse formal, in der Tiefe und kontinuierlich beteiligt werden. Royce schlägt zusätzlich zur Beteiligung an den Anforderungen vor, den Kunden nach dem vorläufigen Design, während der wirklichen Designphase permanent und während der Testphase ebenfalls hinzu zu ziehen. Einsicht, Urteilsvermögen und Zustimmung des Kunden könnten die Entwicklungsanstrengungen nachhaltig unterstützen.

Royce schließt seine fünf Punkte mit einem Bild ab, dass nur noch im entfernten an einen Wasserfall erinnert und schon eher Eschers Wasserfall ähnelt, der die Gesetze der Schwerkraft außer acht lässt.

Schon an dieser Stelle und ohne explizite Untersuchung ist deutlich, dass Royces Interventionen in die Richtung Agilität weisen.

5. Agilität nach Royce

Winston Royce Sohn Walker Royce schreibt über seinen Vater: "He was always a proponent of iterative, incremental, evolutionary development. His paper describes the waterfall as the simplest description, but that it would not work for all but the most straightforward projects. The rest of his paper describes [iterative practices] within the context of the 60s/70s government-contracting models." (Royce nach [Lar 03, 48])

Royce, Winston Royce, beschriebe also iterative Praktiken im Kontext der Softwareentwicklung der 60er und 70er Jahre, wie sie für die US-Regierung zu erfolgen hatte. Wenn wir versuchen ihm gerecht zu werden, müssen wir seine Intentionen aus ihrem Kontext schälen. Es ist nicht entscheidend welche Methode, welche Technik Royce vorschlägt, sondern welche Verschiebung er vornimmt, was die Intention seines jeweiligen Vorschlages ist.

Kehren wir noch einmal zu seinem Aufsatz zurück. Das erste Bild des Artikels beschreibt nicht den Wasserfall, sondern eine kleinere Szene. Es handelt sich um zwei Boxen, natürlich in Treppenform, mit einem Pfeil von oben nach unten und den Beschriftungen "Analysis" und "Coding". [Roy 70, 328] Royce beschreibt diese Grafik, indem er auf zwei Schritte verweist, die für jedes Computerprogramm - egal welcher Größe, egal welcher Komplexität - essentiell sind: Analyse und Programmierung. Diese Schritte seien alles was erforderlich sei, wenn das zu erstellende Programm hinreichend klein sei und diejenigen, die es bauten, es auch benutzen würden. Und - so fährt er fort - für diese Schritte würde auch gern jeder Kunde bezahlen, weil die Schritte unmittelbar kreativ seien und direkt zum Nutzen des Endproduktes beitrügen. (Ebenda) Größere Systeme so zu entwickeln müsse scheitern, deshalb würden viele Phasen hinzugefügt, die keinen unmittelbaren Gewinn brächten wie Analyse und Programmierung.

Mit diesen Gedanken leitet Royce seine Ausführungen ein. Ein unmittelbares - in einem modernen Sinne schlankes (lean) - Entwicklungsmodell habe seine Grenze und müsse um weitere Phasen ergänzt werden. Royce Expansionsmodell ist extensiv. Er erweitert die zwei Phasen mit weiteren Phasen, die den großen Prozess handhabbar machen sollen. Zu fragen ist - und damit kommen wir endlich an der Stelle an, an der dieser Text seine These formuliert - ob nicht, ausgehend von Royce Prämisse der Schlankheit des einfachen Entwicklungsmodells, seine extensive Erweiterung in Form einer intensiven Erweiterung aufgehoben werden kann. Gibt es nicht - so würde die Frage an jede Ergänzung lauten - ein qualitatives Element, mit dem der schlanke Prozesse verbessert werden kann, so dass die extensive Erweiterung mindestens eine intensive Alternative bekäme.

Gehen wir seine Erweiterungen noch einmal durch, dieses Mal mit Blick auf seine Intention. Die erste Ergänzung, "Program Design comes first" (ebenda, S. 331), installiert eine vorläufige Designphase vor der Analyse, um eine Intuition für das Ganze zu etablieren. Das bedeutet für das Phasenmodell, dass eine Phase schon das Ganze als Miniatur enthält. Konsequenterweise lässt sich das gar nicht mehr als Phase ausdrücken, hier wird der Gedanke der Iteration in den Raum der Linearität eingeführt. Für das

schlanke Modell bedeutet das, dass die Schritte von Analyse und Programm wiederholt werden müssen, dass vor der Analyse schon das Programm steht.

In vielen agilen Konzepten ist dieser Gedanke, ein früher Entwurf, der eine Intuition für das Ganze entstehen lässt, aufgehoben. Allerdings wird die Intuition in agilen Kontexten nicht nur durch ein vorläufiges Design, sondern auch durch eine vorläufige Implementierung geschaffen. Wir werden darauf zurückkommen.

"Document the Design." Dieser Grundsatz Royces steht zunächst im Gegensatz zu dem agilen Wert der funktionierenden Software: "Working software over comprehensive documentation" (Agiles Manifest) Für Royce ist die Dokumentation eine Schlüsselfrage. Er folgt seinem Grundsatz der extensiven Erweiterung, indem er sechs Dokumente benennt, die verbindlich in verschiedenen Phasen des Prozesses erstellt werden müssen. Was ist Royce Grund für diese Rolle der Dokumentation? Sie sei das entscheidende Medium der Kommunikation im Prozess. Wenn wir die Kommunikation als den Wert betrachten, der hinter der Dokumentation steht und die Dokumentation als die Technik, die den Wert etabliert, ist diese Technik notwendig, weil der Prozess als arbeitsteiliges Phasenmodell gedacht wird. In Royce Beschreibung des schlanken Modells spielt Dokumentation keine Rolle, weil zwei Phasen immer in einem unmittelbaren Verhältnis stehen und die Entwickler auch die Anwender sind, es also ein Kommunikationsproblem Kunde-Entwickler/Designer nicht geben kann.

Agile Software Entwicklung teilt diese Einsicht mit Royce. Kommunikation ist das Schlüsselproblem. Anders als Royce, der nur die Möglichkeit der Entkoppelung von Phasen und Rollen sieht und damit einer formaleren Form der Kommunikation wie der Dokumentation das Wort reden muss, versucht die agile Softwareentwicklung Techniken zu etablieren, die es erlauben, möglichst lange an Formen der Unmittelbarkeit der Kommunikation festzuhalten. Alistair Cockburn drückt diesen Aspekt der Agilität im Begriff der "osmotischen Kommunikation" [Coc 05, 57ff.] aus. Die Techniken sind unterschiedlich, die Gewichtung der Rolle der Kommunikation ist gleich.

"Do it Twice." Royce setzt mit diesem Grundsatz den Gedanken des vorläufigen Designs fort. Das System soll nicht nur im Vorab en miniature entworfen werden, es gilt auch es einmal zu implementieren. Spätestens diese 'Phase' führt den Begriff der Iteration in das Modell des Wasserfalls ein.[7] Eine der Grundtechniken der agilen Entwicklung, den Prozess von Analyse und Programmierung zu perpetuieren, wird hier in das Modell des Wasserfalls eingebettet. Es lässt sich nicht von einem intensiven Gegenstück zu dieser extensiven Technik sprechen, weil die Technik bei Royce selber schon die Hülle - Linearität des Prozesses - auflöst. Lediglich die Vervielfältigung dieser Operation, der Iteration als Bewegungsform eines Projektes, ist so noch nicht formulierbar. Neben der Form des Kreises, haben viele agile Projektmodelle explizite Begriffe für diese Form des frühen Miniatursystems: Leuchtspurmunition (Tracer Bullits) (vgl. [Hun 03, 44ff.] und [Ric 06, 121ff.]) , das wandelnde Skelett (vgl. [Coc 05, 83f.])

[7] Spätestens jetzt, mit dem zu sich selbst wiederkehrenden Wasserfall, wären wir bei Escher.

"Plan, Control and Monitor Testing." Die Nähe zu agilen Methoden könnte nicht größer sein. In Royce extensiver Denkweise formuliert drückt sich der hohe Stellenwert des Testens in spezifischen Testphasen mit ausdrücklichen Dokumenten aus. Gelingt es Testen zu einem integralen Bestandteil des Entwicklungsprozesses selbst zu machen, gibt es Analyse und Programmierung gar nicht ohne ihre Testdimension, dann kann auf die Erweiterung des Modells mit Testphasen verzichtet werden, denn die Entwicklung bedeutet 'Test in Permanenz'. Mit Akzeptanztest und Unittests sind es gerade die Testdimensionen von Analyse und Programmierung, die auf den Begriff gebracht werden.

Royce Gedanke der visuellen Inspektion, die einen großen Teil der Fehler in der Testphase aufspürt, wird in der agilen Praxis in Form des Pairprogrammings bzw. des kleinen, permanenten Codereviews [Ric 06, 101ff.] aufgehoben.

"Involve the Customer." Royce merkt selber, dass die Kundenbeteiligung nicht als zusätzliche Phase oder zusätzliches Dokument integriert werden kann. Er fordert den Kunden früh und vielschichtig einzubeziehen. "Kunde vor Ort" [Bec 00, 68] lässt sich als Endpunkt des Weges verstehen, den Royce einschlägt.

Ist Royce geläutertes Wasserfallmodell ein agiler Ansatz? Nein, eindeutig nicht. Aber er erweitert das Modell um Dimensionen, die auf Werten fußen, welche auch zentral für das Selbstverständnis agiler Softwareentwicklung sind. Royce, das mag ein Moment seiner Zeit sein, erweitert das zugrundeliegende Modell in einer extensiven Art und Weise. Er fügt hinzu, ergänzt, denkt arbeitsteilig. Agile Softwareentwicklung ergänzt das zugrunde liegende Modell - Analyse und Programmierung - intensiv.[8] Die Elemente werden reichhaltiger, vielfältiger aber nicht vervielfacht. Tests bedeuten nicht eine neue Phase, sondern werden zum Bestandteil der Bestehenden. Die Iteration wird nicht zu einer neuen Phase, sie wird zum Bewegungsmodus des Modells schlechthin.

Royce entwickelte - mit den extensiven Mittel seiner Zeit - das single-pass Wasserfallmodell in eine agile Richtung weiter.

References

[Bec 00] K. Beck: *Extreme Programming. Die revolutionäre Methode für Softwareentwicklung in kleinen Teams.* München. 2000

[Coc 03] Alistair Cockburn: *Agile Software-Entwicklung*. Bonn. 2003 (281)

[Coc 05] Alistair Cockburn: *Crystal Clear. Agile Software-Entwicklung für kleine Teams*. Bonn. 2005

[Hun 03] A. Hunt, D. Thomas: *Der Pragmatische Programmierer.* München, Wien. 2003

[Lar 03] C. Larman, V. R. Basili: Iterative and Incremental Development: A Brief History in: *IEEE Computer.* Juni 2003

[Ric 06] J. R. Richardson, W. A. Gwaltney: *Ship it! Softwareprojekte erfolgreich zum Abschluss bringen.* München, Wien. 2006

[Pop 03] M. Poppendieck, M. Poppendieck: *Lean Software Development. An Agile Toolkit.* Bosten, San Francisco, New York. 2003

[Roy 70] W. R. Royce: Managing the Development of Large Software Systems in: *Proceedings. IEEE WESCON.* August 1970

[8] Auch das ist - natürlich - ein Moment unserer Zeit.

[Sch 96] Ken Schwaber: *SCRUM Development Process. http://*jeffsutherland.com/oopsla/schwapub.pdf
[Wik 06] Wikipedia: *Wasserfallmodell.* http://de.wikipedia.org/wiki/Wasserfallmodell. Zugriff am 10.2.2006