ruynk

Projektwissen

Untersuchung bezüglich der SW-Entwicklungsmethoden in Unternehmen (mit dem Schwerpunkt "Agile")

und ansatzweise Suche einer Lösung für die Unzulänglichkeiten der gegenwärtigen SW-Entwicklungsmodellen

Bearbeitet: Ruy Kuhlmann

Version: Juni 2017

Internet

http://www.ruynk.com
AMTRS: http://www.amtrs.de
Ruy Kuhlmann: http://www.ruynk.de
IT-Wissen Ryusui: http://www.ryusui.de
Tsubame Software Tools: http://www.tsubame.de

Untersuchung bezüglich

der SW-Entwicklungsmethoden

im Unternehmen

mit dem Schwerpunkt "Agil"

und ansatzweise Suche einer Lösung

für die Unzulänglichkeiten

der gegenwärtigen SW-Entwicklungsmodellen

Bearbeitet: Ruy Kuhlmann

Version: 15. Juni 2017

Alte Versionen
3. vom 08. Nov. 2016
2. vom 27. Oktober 2016
1. Draft vom 20. Oktober 2016

Inhaltsverzeichnis

Copyright

Dieses Dokument ist Eigentum von Ruy Kuhlmann.
Es darf auch ohne meiner Genehmigung vervielfältigt und auch dritten Personen zugänglich gemacht werden unter der Voraussetzung, dass „Ruy Kuhlmann / ruynk" als Autor erwähnt wird.

Kontakt über meine Homepage:
http://www.ruynk.com

Einleitung

Diese Untersuchung beinhaltet viele Texte aus Entwicklersicht, von Foren und Blogs übernommen (Übersetzungen von mir: R.K.).

Anmerkungen:
1) Die übernommenen Texte sind in blauer Farbe zu sehen, meine eigene sind einfach in schwarzer Farbe.
2) Die Quelle der Meinungen sind Blogs und Kommentare; diese können am Ende unten „Links" nachgeschlagen werden. Viele sind im Original in englischer Sprache.

Ich befasse mich schon seit dem Jahre 2010 mit den "agilen" Methoden; als die Firma Nimsoft AS, wo ich arbeitete, von der CA aufgekauft und "Scrum" eingeführt wurde. Seitdem verfolge ich die agilen Methoden und lerne über ihre Eigenschaften und Entwicklung.

Ich bin nicht immer und nicht unbedingt von den hier wiedergegebenen Meinungen überzeugt: Überzeugt bin ich davon, dass Positionen relativ sind und eine Äußerung oft nur zu einem bestimmten Zeitpunk an einem bestimmten Ort und zu einem bestimmten Zusammenhang anwendbar ist.

Auch meine Stellungnahmen (dort wo es so steht, ist es auch meine Meinung) sind eher als Gedankenanstoß zu verstehen und nicht als die absolute, dogmatische Wahrheit (welche ich nicht gepachtet habe).

An die Leserinnen hier: ich habe durchgehend männliche Endungen verwendet ohne Rücksicht auf die weiblichen. Das dient der Lesbarkeit; **eine besondere Behandlung jeglicher Ausprägung ist nicht beabsichtigt**.

Es kann durchaus sein, dass hier Texte sich wiederholen bzw. dass sehr ähnliche Sätze (also von verschiedenen Quellen wahrscheinlich) zu lesen sind: Das ist jedoch Absicht: Wiederholungen sollen das Gewicht der Gedanken betonen.

Die Entscheidung, ob eine Aussage hier aufgenommen werden sollte oder nicht, fiel mir nicht immer leicht. Sicherlich habe ich einige interessanten Texte weggelassen und andere unwichtigen irrtümlicherweise hinzugefügt. Wenn Sie als Leser etwas finden, was Sie nicht weiterbringt, so können Sie es mir gerne schreiben.

Dieses Buch ist im Laufe mehreren Monaten entstanden, daher finden Sie im Laufe dieser Untersuchung auch Passagen, die zwar nicht wirklich zu anderen Ideen passen, die jedoch später zu einem Teil der Lösung wurden; ersteren habe ich aus historischen Gründen belassen, um die Umwege zu einer Lösung auch mit zu dokumentieren.

Und jetzt, noch vor dem Haupttext, möchte ich mich in aller Form entschuldigen für sämtliche Aussagen, wo die "agilen" Methoden eine schlechte (bis sehr schlechte) Bewertung erhalten haben. Leider konnte ich, weil die Schilderung so zutreffend war oder sogar aufgrund der gebotenen Aufrichtigkeit, einige (nicht genehmen aus agiler Sicht) Sätze nicht wegstreichen. Sie sind nun erhalten geblieben. Dafür bitte ich im Voraus um Verzeihung.

An dieser Stelle will ich mich bei allen den Kollegen bedanken, bei denen ich zusammenarbeiten durfte und bei den Firmen, wo ich tätig sein durfte.

Zweck dieser Untersuchung ist uns allen, (Entwickler, Projektmitarbeiter, Management etc.), nachdenken lassen über das, was wir gerade machen, insbesondere im Hinblick auf die sogenannten „agilen“ Methoden.

Grundsatz

Um gute Programme zu entwickeln braucht man grundsätzlich 2 Punkte zu beachten:
1. Einverständnis über das, was das Programm tun soll, und
2. Ein ausgeklügeltes Programm schreiben, das das tut
Es ist wichtig sich im Klaren zu sein, dass alle beide Aufgaben schwer sind, nicht nur die zweite.

Das Schreiben einer guten, präzisen und detaillierten Spezifikation für ein komplexes System ist ein wirklich harter Job; in Wirklichkeit schwerer als das Programmieren selbst. In letzter Instanz ist das Schreiben von Programmen nur eine sehr ausführliche, lauffähige Spezifikation des Problems.

[Anmerkung R.K.: Ich würde eher 'eine sehr ausführliche, lauffähige Abbildung von Ausgängen als Funktion der Eingänge' schreiben.]

Allgemeine Begrifflichkeiten

Modell und Methode: Einem Modell soll eine Modellierung zugrundelegen, ein Bild oder Zeichnung ohne wesentliche Änderungen oder Anpassungen. Eine Methode im Gegenteil beschreibt eine Vorgehensweise und ist, somit, Änderungen unterworfen, wenngleich diese nicht all zu oft vorkommen sollten. Daher würde ich das Wort 'Modell' auf wohldefinierte Methoden wie CMMI, RUP, SPICE usw. und das Wort 'Methode' für die agilen anwenden, die sich eher aus einem Sammelsurium an Praktiken zusammensetzen, die man nehmen kann oder lassen kann, je nach Laune.

Validieren (VAL): Hat den Zweck zu zeigen, dass ein Produkt die ihm zugedachte Verwendung erfüllt.

Verifizieren (VER): Hat den Zweck sicherzustellen, dass die Ergebnisse der Arbeit die Anforderungen erfüllt.

Über Brainstorming; funktioniert nur in kleinen Gruppen. Und das überrascht nicht wirklich.

Es dürfte klar sein, dass Management nicht gleich Management ist. Es gibt, wie bei allen Gegenständen der Arbeitswelt, gutes, mittelmäßiges und schlechtes Management. Im gesamten Text kommen natürlich alle drei vor; um sie nicht gleich alle in einem Topf zu werfen (das würde erstens zu Missverständnissen führen und zweitens mich verständlicherweise unbeliebt – noch unbeliebter - machen) bin ich dazu übergegangen, das *unerwünschte Management* mit Hochkomma zu kennzeichnen, folgendermaßen: 'Management'.

Englisch - Deutsch

Unter anderem versuche ich hier auch, den Wildwuchs an englischen Begriffen durch deutsche Begriffe zu ersetzen. Das hilft auch oft dabei, Buzzwörter zu identifizieren. "Buzzwort" selbst wird als "Schlagwort" übersetzt, ist aber eher als Propaganda-Wort oder Reklame-Wort zu verstehen. Bei den Übersetzungen aber habe ich teilweise das englische Wort weiterhin verwendet, um die Aussage so nah wie möglich an dem Original zu belassen.

RE = Requirement Engineer; ist ein Anforderungsexpert

PO = Product Owner ist ein Produkteigentümer. Der Begriff ist irreführend, der Kollege ist nicht der Eigentümer, sondern er vertritt nur die Sicht des Kunden bei einer Entwicklung. Spielt also die Rolle einer Schnittstelle zwischen Kunde und Entwickler. Manchmal wird er auch als "Proxy" bezeichnet.

RE wurde zu PO (Requirement Engineer wurde zu „Product Owner“)

Skill = Fähigkeit oder Können. Ich möchte es als "Wissen und Können" beschreiben.

Manager = Führer oder Leiter, je nachdem wie stark er seine Position vertritt. Ein Manager also führt oder leitet. Doch was ist Management? Das dürfte die Leitung eines Unternehmens oder Unternehmung sein.
Leitung eher als Führung eines Unternehmens, nach heutiger Auffassung (2016). Ein "Geschäftsführer" wäre demnach eher ein "Geschäftsleiter".

To write Code = Programmieren: Ich halte die Übersetzung "Code schreiben" für falsch, denn es geht beim Programmieren um weitaus mehr, als einfach Linien in einer Programmiersprache zu schreiben.

Issue = Abweichendes, zu untersuchendes Ereignis. (Auch als Falschzustand bzw. Problem zu verstehen).

Incident = Ein Vorkommen in dem Sinne von etwas Unerwartetes, gut oder schlecht (oft aber schlecht).

Good Practice = Richtige, bewährte Vorgehensweise

Stakeholder = Teilhaber (an einem Projekt normalerweise)

Lean = Schlank

Das hier trifft nur für die englischen Texte; wir in Deutschland benutzen richtig das Wort "Methode" und keine "Methodologie". Zu beachten aber wenn Sie

englische Texte lesen:

"Hate how the word 'methodology' has been highjacked to make following a 'method' sound more scientific.
If you are following a procedure, or talking about a defined procedure it is a METHOD.
If you are talking about the study of methods then it is 'methodology'. Basic High School English, if you add '-ology' to the end of a word you are talking about the study of."

Erfahrung – Wie die Entwickler ihre Welt sehen

Die Arbeit an sich, das Erstellen des "Dinges", wird von jedem einzelnen Mitarbeiter an seinem Arbeitsplatz bewerkstelligt. Alles andere ist Verschwendung. Außer, der Mitarbeiter ruht sich aus, trinkt Kaffee, unterhält sich mit Kollegen, lernt, plant, denkt. Diese Aktivitäten jedoch sollen aus eigenem Antrieb erfolgen; nicht in Meetings oder Kreisen oder Boards oder sonst wo "gepuscht" werden. Jedes "Puschen", jeder Zwang ist bei Weitem nicht so effektiv wie das Schaffen aus eigenem Antrieb.

Ist das Designen Tot? - von Martin Fowler (Mai 2004)
"Geplantes Design zählt dazu und enthält einen Begriff, der aus anderen Branchen der Ingenieurwissenschaft stammt. Wenn Sie eine Hundehütte bauen wollen, können Sie einfach etwas Holz zusammen nageln und gut ist es. Allerdings, wenn Sie einen Wolkenkratzer bauen möchten, können Sie nicht auf diese Weise vorgehen - es wird zusammenbrechen, bevor Sie sogar auf halber Höhe gekommen sind.
[...]
Designer wurden zu Designer aufgrund Wissen und Erfahrung.
[...]
Geänderte Anforderungen sind das größte Problem, das mir in Software-Projekten bekannt ist.

Eine Gemeinsamkeit ist mir aufgefallen, es scheint so zu sein, dass Eigentümer (Manager, Stakeholder usw.) nicht in der Lage sind das zu dokumentieren (kommunizieren), was sie wollen, oft wissen sie nicht mal, was sie wollen.

[Anmerkung R.K.: Zu beachten bei einem Lösungsansatz.]

[Anmerkung R.K.: Der agile Ansatz wäre demnach nur konsequent: Wenn der Kunde nicht dokumentieren kann, was er will, müssen die Entwickler auch nicht das dokumentieren, was sie tun.]

Das Ziel eines jeden Prozesses ist es Programmierer in der Lage zu versetzen, guten Code zu schreiben (gut zu programmieren).

[Anmerkung R.K.: Mit “Prozess“ ist wahrscheinlich eine Entwicklungsmethode oder -modell gemeint.]

Etwa die Hälfte aller Software-Entwickler haben Informatik studiert, viele andere haben Mathematik oder Physik studiert, also wurden viele der Entwickler von naturwissenschaftlichen Universitäten ausgebildet. Doch die naturwissenschaftlichen Universitäten trainieren die Menschen nicht in der Produktentwicklung. Es gibt keine Kurse in Entwicklungsmetriken, Konfigurationsmanagement, Projektmanagement oder Qualitätssicherung. Grundsätzlich ist die überwiegende Mehrheit der Mitarbeiter, die heute Software produzieren, nie durch ihre Ausbildung für diese Rolle vorbereitet. Sie haben sich im Wesentlichen alles selbst beigebracht. Und sie wissen nicht, was sie nicht wissen!

Eine Antwort ist einfach: das "Problem" entsteht nicht wegen einer Methode, das "Problem" sind die Skills der Mitarbeiter... Neue Entwicklungsmethoden werden erfunden um die lahme Entwicklung zu kaschieren, anstatt das Übel direkt an der Wurzel zu packen: schlechte Skills. Wirklich gute Entwickler werden sogar mit Wasserfall oder was auch immer erfolgreich sein. Der Punkt ist der, dass die Ausbildung nicht leicht ist, sie erfordert viel Zeit und Arbeit, so dass sie am liebsten ganz eingespart werden soll.

[Anmerkung R.K.: Es geht hier um die “nackte“ Arbeit, das wahre Ding.]

Das strenge Management, die Gantt-Diagrammen, welche Zeitpläne und Dokumentation liebte, wurde zu "Stakeholdern" und "Benutzer" reduziert, sodann in "User Stories" weg abstrahiert. Für mich ist es nun zur Gewohnheit geworden, in Projekten zu arbeiten, die anscheinend keine Aufsicht durch Erwachsene haben.
Überraschenderweise verwandeln sich die sich selbst überlassen Entwickler unter diesen Umständen nicht zu wilden "Code-Hacker"... Sie übernehmen oder erstellen vielmehr eigene Methoden welche noch strenger sind, und führen Rituale ein die alles, was ich im Jahr 1980 erlebt hatte, in den Schatten stellen.
Tatsächlich können Entwickler heute viel rigider und dogmatischer über die angewendeten Methoden sein, als man sich vorstellen kann, dass eine COBOL-Gruppe in der 1970er war.
Ich arbeite zurzeit immer wieder bei Projekten, bei denen einer oder zwei Mitglieder sich so sehr mit Prozessen und "Best Practices" befassen, dass so

gut wie nichts mehr von wirklichem Wert erzeugt wird.

Ob eine Entwicklungsmethode funktioniert oder nicht, hängt eher von folgenden Kriterien ab:
Teamproduktivität, Spaß, Wertschätzung, Konformität der SW, Vorhersehbarkeit der Arbeit, Verantwortlichkeit (bzw. Verantwortungsgefühl), Kommunikation unter den Stakeholder, geschriebene Programmlinien pro Tag, Mann-Monate (genug Entwickler also), Code-Qualität, produzierte Artefakte, usw.
Jede Methode funktioniert, wenn Sie das Richtige messen. Aber in Bezug auf die einzige Messung, die wirklich wichtig ist: "Das Programm deckt die Anforderungen innerhalb der Zeit und der Kosten", habe ich noch keine Methode gesehen, die konsistent Ergebnisse liefert.

[Anmerkung R.K.: Daraus könnte ich folgen, dass nicht die "Methode" an sich funktioniert, sondern die Mitarbeiter.]

[Anmerkung R.K.: Das Ergebnis ist aus meiner Sicht nicht wirklich zu messen; entweder geschafft oder nicht. Es sind einzelne Parameter im Laufe des Projektes, die hierüber Auskunft geben können und sollen.]

Brooks hatte Recht, es gibt keinen Königsweg (silver bullet).
Was ich funktionieren gesehen habe:
1) Setzen Sie die kleinste Anzahl der bestmöglichen Mitarbeiter zusammen.
2) Geben Sie ihnen ein klares Ziel.
3) Lassen Sie sie gewähren.

Ich denke, Programmierer sollen viel mehr auf die Arbeit fokussieren und mit den Stakeholder zu kommunizieren, als sich mit Ritualen und Tools zu beschäftigen. Darüber hinaus sollten wir alle recht skeptisch sein, wenn bestimmte Prozesse überhand gewinnen oder Methoden versprechen, auf magischer Art und Weise alle Mitarbeiter produktiver zu machen.

Ich mache mich Sorgen, weil die Gesamte Software-Entwicklung, durch eine allgemeine Verspieltheit, den Fokus verschoben hat; weg von der Qualität und hin zu unkritischen Zielen. Was am Ende wirklich zählt, ist die Qualität des Systems und ob es tatsächlich das Leben und die Arbeit der Benutzer leichter oder besser macht. Alles andere ist zweitrangig.

Letztendlich verplempern die Entwickler ca. 50% der Zeit damit, Aufgaben zu verwalten, und nicht damit, sie auszuführen.

[Anmerkung R.K.: Damit leidet in besonderem Maße auch die Qualität der Ergebnisse.]

Folgegedanken

Vielleicht weil so viele "Fremde" bei der Entwicklung aufgenommen worden sind, die das Handwerk (Programmieren) nicht wirklich beherrschen, verpulvert man so viel Zeit mit "Ritualen" anstatt die echte Arbeit zu verrichten: Weg von der Ingenieur-Mentalität und hin zur geisteswissenschaftlichen Arbeit.

Über die Entwicklung

Heutzutage blasen Programmierer die Komplexität ihrer Arbeit künstlich auf, unter dem Vorwand der "Best Practices", in der Hoffnung die Arbeitsplatzsicherheit zu steigern. Und vielleicht wollen sie auch, dass andere Mitarbeiter sie als "Experten" in dem Gebiet wahrnehmen.

Wie auch immer, zum Glück hörte ich als Angestellter mit dem Programmieren zum richtigen Zeitpunkt auf, als dieser Horror [N.B.: gemeint sind die "Agilen"] Einzug hielt. Nun kann ich meinen Kaffee unbehelligt trinken und mitten in der Nacht programmieren, um 11:00 oder 12:00 Uhr aufstehen, so wie Gott wollte, dass die Software-Entwicklern es tun.

Eine ganze Industrie und Dutzende von Institutionen arbeiten an Mitarbeitermotivation und Spielchen spielen und Aktivitäten durchführen usw. usf... und Unternehmen kaufen Systeme, stellen "gutes-Gefühl" Manager ein, gestalten lustige Büros mit viel Unterhaltung... doch das Wichtigste wird vergessen: Das Ziel treibt die Mitarbeiter. (Siehe hierzu unter "Mythen" den Punkt **Arbeitermotivation**).

Lehnen Sie ab oder passen Sie Technologien an, wenn Konflikte mit Ihrer Firmen-Kultur, oder mit Stabilität, Zuverlässigkeit und Vorhersagbarkeit zu befürchten sind.

Inversion of Control (IoC = Umkehrung der Kontrolle)

Bevor Sie "Inversion of Control" verwenden, sollten Sie sich der Tatsache bewusst sein, dass sie Vor- und Nachteile hat, und Sie sollten wissen, warum Sie sie verwenden, wenn Sie dies tun.

Vorteile:
- Ihr Code wird entkoppelt, so kann man leicht Implementierungen einer Schnittstelle mit alternativen Implementierungen austauschen
- Sie zwingt regelrecht zur Codierung gegen Schnittstellen anstelle von Implementierungen
- Es ist sehr einfach Unit-Tests fur den Code zu schreiben, weil es nur die Objekte in seinem Konstruktor bzw. Setter akzeptiert und man kann sie leicht mit den richtigen Objekten isoliert initialisieren.

Nachteile:
- IoC kehrt nicht nur der Steuerfluss im Programm um, es vernebelt es auch beträchtlich. Das heißt, es geht nicht mehr den Code einfach zu lesen und von einer Stelle zur anderen zu springen, weil die Verbindungen zwischen dem Code nicht mehr im Code sind! Stattdessen ist es in XML-Konfigurationsdateien oder Anmerkungen und diese Metadaten werden "interpretiert".
- Es entstehen neue Fehlern, wenn lediglich eine XML-Konfiguration falsch ist; um herauszufinden, was Ihr IoC-Container in einem Objekt unter bestimmten Bedingungen injiziert, kann man eine lange Zeit (sehr lange) verbringen.

Ich persönlich sehe die Stärken von IoC und mag sie wirklich, aber ich neige dazu, IoC zu vermeiden, wann immer möglich, weil sie eine Software zu einer Sammlung von Klassen verwandelt, die nicht mehr ein "echtes" Programm ist, sondern nur ein "Etwas", das irgendwie zusammengebunden durch XML-Konfiguration oder Metadaten werden muss. Diese lose Klassen würden, ohne die Metadaten, auseinander fallen.

Fazit aus meiner Sicht

Dieser Text über IoC hat wenig mit den agilen Methoden zu tun, ich habe es jedoch aufgenommen weil die IoC eine Verkomplizierung der SW-Entwicklung mit sich bringt.

Seit der Zeit der "glorreichen" Einführung des OOP als Lösung aller Entwicklungsübel (Anfang der 90er), scheint die Entwicklerwelt aber immer noch verzweifelt nach Lösungen zu suchen und keine finden zu können.

Über die Entwicklungs-Methoden und -Modellen

Ich bin nicht der Überzeugung, dass eine gegebene Entwicklungs-Methode an und für sich ein vorhersehbares und/oder wiederholbares Software-Entwicklung-Prozess liefert (N.B.: Gemeint, denke ich, ist „Ergebnis“).

Man unterliegt gerne dem Irrtum zu glauben, dass ein Modell ein Kochbuch gleicht und dass kein (weiteres) Denken zum Einsetzen erforderlich ist. Zu oft habe ich Leute sagen gehört; "folgen Sie nur diese Methode und es wird einfach funktionieren", ohne zu verstehen, dass Methoden nur Rahmenbedingungen sind, die auf das Problem angepasst werden müssen.

Das Problem ist die Lücke in der Struktur der Organisation; zwischen den Manager die sagen was sie wollen und die Entwickler, die das erstellen sollen: Die Schicht dazwischen. Die Mitarbeiter, die Anforderungen in eine Entwicklungsstrategie "übersetzten", die Objekte und u.a. auch die Programme planten (die Leute also die aus PowerPoints ein Programmdesign erstellten), sind weg.

Ich habe bei großen Projekten gearbeitet, kleinen Projekte, in großen Teams und auch manchmal ganz alleine, bei versteinerten Bundesbehörden und bei "coolen" Unternehmen in Silicon Valley. Ich habe mindestens zwanzig Programmiersprachen gelernt und benutzt. Ich habe mit Wasserfall/BDUF, strukturierte Programmierung, Top-Down, Bottom-Up, modulares Design, "Komponenten", Scrum, XP, TDD, OOP, Rapid Prototyping, RAD und wahrscheinlich andere, die ich vergessen habe, gearbeitet. Ich bin nicht überzeugt, dass irgendeinem von diesen Ansätzen wirklich funktioniert.

[Anmerkung R.K.: Was funktioniert dann in der Wirklichkeit? Ein Team? Einzelne Menschen mit ihren "Skills"? Was ist der Schlüssel zum erfolgreichen Projekt?]

Ich bin überzeugt, Programmierer sind eine Gruppe, bei der die Definition von Wahnsinn gut passt: die gleiche Sache immer und immer wieder machen und dabei andere Ergebnisse erwarten. Ja, die Methoden sind unterschiedlich, aber das "dieses Mal werden wir durch die Verwendung einer Methode die Arbeit zum Laufen bringen"-Mantra ist immer das Gleiche.

Wenn Sie ein gutes Team haben, dann werden die Mitarbeiter auch miteinander reden; jede zusätzliche Zwangs-Kommunikation wird als Belastung empfunden.

Auf der anderen Seite, wenn Teams nicht wirklich oder gar einigermaßen funktionieren, so zwingen die (agilen) Methoden eine Kommunikation, und dies erhöht die Chancen, dass das Projekt erfolgreich wird.

[Anmerkung R.K.: Das hier ist ein wichtiger Punkt zu berücksichtigen, aus dem sich aber ein Paradoxon ergibt: Schlechte Teams werden durch Kommunikation verbessert, gute Teams verschlechtert durch Zwang und erhöhte Verwaltung.]

[Anmerkung R.K.: Nicht wirklich klar geht daraus hervor, ob das (ursprüngliche) Problem die schlechte Kommunikation oder die schlechten Teams (die schlecht kommunizieren) sind.]

Sobald ein Programmierteam eine Entwicklungsmethode angenommen hat, es ist fast unvermeidlich, dass einige Mitglieder des Teams, oder vielleicht auch nur eine Person, die strikte Einhaltung fördert und sie zu einer Religion erhebt. Die sich daraus ergebenden (passiven) Angriffe vernichten die Produktivität schneller als sonst jede technische oder nicht technische Entscheidung.

Ich habe immer die "Entwicklungsmethoden" als eine Möglichkeit begriffen, damit alle (am Meisten aber das Management) ihr Gehirn ausschalten und dabei denken können, dass der "Prozess" die Ergebnisse liefert und nicht der Einsatz des Teams.

[Anmerkung R.K.: Aus meiner Sicht, viel "Einsatz" vom Team ohne eine einigermaßen funktionierende Leitung erzeugt viel Arbeit aber wenig Inhalt, viel Produktivität für wenig Produkt, viel Mühe für wenig Ergebnis.]

Wenn man sich die Methoden anschaut, geht es in erster Linie um Kommunikation. Der Wasserfall Modell versucht, alles auf Papier zu bekommen, so dass es offensichtlich wird. Die agile Modellen mit "Stand-Ups" oder Kanban versuchen nichts anderes als die Kommunikation zwischen den Entwicklern zu forcieren und dafür zu sorgen, dass jeder "im Boot" ist.

Alle Software-Methoden sind ein verzweifelter Versuch, unfähige Mitarbeiter am Spiel Teil nehmen zu lassen. Von Scrum zu TQM, alle sind Versuche, Menschen mit einem sozialwissenschaftlichen Studium einzubeziehen. Diese Leute sind billige Angestellten und das Management ist überzeugt, dass es ausreicht, wenn genug Unkundigen (nur) einem Programmierer sagen, was er zu tun hat.

[Anmerkung R.K.: Ich denke nicht, dass die Progammierfremden billige Leute sind. Das Problem scheint eher eins wie das mit dem Ei und dem Huhn zu sein: Weil man nicht genug Entwickler fand, hat man auf Menschen gesetzt, die sich

zwar selbst mit Hilfe eines Personal-Computers eine Menge beigebracht haben, doch im Programmierhandwerk eher rein pfuschen, und um diesen Missstand Herr zu werden, wurden zuerst OOP, dann Design-Patterns, dann agile Methoden eingeführt. Man darf gespannt sein, was als nächstes kommt.]

Wie auch die Design-Patterns, die meisten Software-Entwicklungsmethoden sind Lösungen auf der Suche nach Problemen.

Aus dem Zustand heraus, mit lustigen Begriffen um sich zu werfen oder aus neuen Perspektiven Altes zu beleuchten sind Software-Entwicklungsmethoden niemals herausgewachsen.
Gute Software ist das Ergebnis von fähigen Mitarbeiter, welche entweder miteinander oder ganz alleine mit dem Ziel arbeiten, qualitativ hochwertige SW zu schreiben.

Die IT-Berater sollen einen großen Teil der Verantwortung für das Unbrauchbar-Machen der Methoden übernehmen. Nicht lediglich oft, sondern immer ist es in ihrem Interesse ein Projekt mit zusätzlichen Mitarbeiter zu bestücken, Prozesse aufzublähen usw. damit sie, die Projekte, so lange wie möglich bestehen bleiben. Ihre Version (die der IT-Berater) von "Good Practice" hat nur Profit als Ziel... und das funktioniert erschreckend gut.

[Anmerkung R.K.: Mehr Mitarbeiter bedeuten mehr Verwaltungsarbeit. Ab einer bestimmten Größe sind die Mitarbeiter hauptsächlich damit beschäftigt, Verwaltungsarbeiten durchzuführen.]

[Anmerkung R.K.: Ich denke, die IT-Berater haben die "Methoden" nicht unbrauchbar gemacht, sondern lediglich, wie es zu erwarten war, missbraucht um ihr Job anzubieten (ihre Leute im Projekt unterzubringen, ihr Einkommen zu maximieren etc.).]

Fazit aus meiner Sicht

Entwicklungsmethoden bzw. -Modellen sind nicht immer eine Lösung sondern oft ein Teil des Problems.
Die eine oder andere Methode adressiert bestimmte Facetten der Schwierigkeiten bei der SW-Entwicklung, doch die herangezogenen Lösungen lassen andere Probleme entstehen, so dass unterm Strich nichts gewonnen, ja bisweilen sogar verloren wurde.

Über das sogenannte Wasserfall Modell

Sehr selten habe ich eine genaue Beschreibung des sogenannten Wasserfall-Modells gesehen. Die Befürworter aller anderen Methoden benutzen den Wasserfall als Strohmann (Buhmann) um zu zeigen, wie viel besser die jeweilige Methode ist. Aber das Wasserfall-Modell beschreibt lediglich: Anforderungen vor Design, Design vor Implementierung, Implementierung vor Test. Alle Produktionsprozesse folgen im großen und ganzen diesem Modell (wenngleich die TDD-Anhänger dies umkehren zu wollen scheinen). Grundsätzlich aber haben die meisten Mitarbeiter, die in der Software-Entwicklung tätig sind, doch nie wirklich etwas über SW-Entwicklungsmethoden studiert, und so übernehmen die meisten Gruppen einfach eine bestimmte Methode als Ergebnis aus der Vorliebe der lauteste Person in der Gruppe.

[Anmerkung R.K.: Und die lauteste Person ist leider oft genug auch die ungebildetste.]

Das Wort "Wasserfall" ist oft schlichtweg falsch benutzt: Heute benutzt man das Wort "Wasserfall" locker (ungenau) um alle Methoden zu bezeichnen, die einen Plan zu Grunde haben (plangetrieben), und nicht wirklich die genaue Definition vom "Wasserfall", wie sie ursprünglich von Winston Royce im Jahre 1970 benutzt worden ist.

Iterative Methoden z.B. wie Rational Unified Process (RUP) samt Variationen davon wurden ziemlich weit in den 1990er und frühen 2000er Jahren verwendet, und viele Menschen heute würden das "Wasserfall" nennen, weil sie gewissermaßen nach Plan arbeiten; doch diese Methoden passen überhaupt nicht in die klassische Definition vom "Wasserfall".

Der klassische Vergleich von "Agil gegen Wasserfall" geht aus einer Menge von Stereotypen hervor, von Mythen und Missverständnissen über beide Methoden, sowohl Agile als auch Wasserfall, und es ist schlicht eine falsche Dichotomie, dass es sich um eine "binäre" bzw. eine "sich gegenseitig ausschließende" Wahl zwischen den beiden handelt. Eine objektive Betrachtungsweise ist es, an einem Ende die Methoden mit starkem plangesteuerten Ansätze und am anderen Ende die mit starken anpassungsfähigen (sogenannte "agilen") Ansätze zu platzieren, dazwischen existieren ja sehr viele Alternativen.

[Anmerkung R.K.: Das Gegenteil von "geplant" ist "ungeplant" oder "ohne Planung". Das Gegenteil von "anpassungsfähig" (Flexibel) ist "starr" oder "rigide".
Vereinfacht:

Geplant => Starr
Ungeplant => Flexibel
Weiter:
Planung (sich Gedanken machen) => Starr
Keine Planung (keine Gedanken) => Anpassungsfähig
Folglich:
Eine ungeplante Rigidität (ist Scrum samt Ritualen... nicht eigentlich eine ungeplante Rigidität?) und, last but not least,
eine geplante Flexibilität wären noch zu untersuchen.]

Der Ansatz, der als "Wasserfall" bekannt wurde, wurde zuerst von W. W. Royce in einem Schreiben vom Jahre 1970, "Managing the Development of Large Software Systems", beschrieben. Darin erklärt Royce, dass es zwar großartig wäre, zuerst Systemanforderungen zu definieren, sodann SW-Anforderungen daraus zu entwickeln, dann diese zu analysieren, anschließend ein Programm zu designen und erst dann zu entwickeln, am Ende schön testen und in Betrieb nehmen. Und auch wenn man stark erwarten würde, dass man nur zwischen den Schritten zu iterieren müsste, dennoch wäre so ein Ansatz riskant und fehleranfällig. Das hat er geschrieben.

Das Risiko ist, so Royce:
Erst bei der Testphase, die doch am Ende der Entwicklungsphase stattfindet, werden abweichende Verhaltensweise festgestellt. Diese Abweichungen lassen sich nicht einfach (genau) analysieren (nicht in dieser Phase). Sollte auch noch das System mehrere externen Anforderungen nicht gerecht werden, so wäre ein großes Redesign fällig: Entweder soll man die Anforderungen ändern oder das System müsste in wesentlichen bzw. großen Teilen neu erstellt werden. Das bedeutet, dass die Entwicklung sich tatsächlich zu dem Startpunkt zurückbewegt hat. An diesem Punkt angelangt ist schon mit einer Überschreitung von 100% Zeit und Kosten zu rechnen.

[Anmerkung R.K.: Und wir haben es hier wieder mit dem Problem der Qualität, damals einfach als "Testen" begriffen, welches schon damals nicht so recht *im Laufe eines Projektes* einbezogen (eingebaut) werden konnte.]

[Anmerkung R.K.: Qualität ist nicht "Testen": Qualität soll und muss von Anfang an im Produkt einfließen. Dieses "Fließen" wäre wie bei der Autoserienproduktion von der Firma Toyota beschrieben ist, zu verstehen.]

Fazit aus meiner Sicht

Ein "Wasserfall" als solches gab es nie in der Wirklichkeit, nur als Theorie in

Royces Schreiben.

Und die "Agilisten" (unter viele Anderen) argumentieren seit Jahren mit diesem "Strohmann" !

Eine Einführung in "Agiles"

"Agil" ist ein Modewort, ein Buzzwort (Reklamewort), ein "Hype".

Eine ganz andere, nicht auf die Software- und Produktentwicklung sondern die Führung sozialer Systeme in risikoreichen und komplexen Missionen ausgerichtete Sicht von "agil" findet sich in "Power to the Edge", einer Publikation des "Command and Control Research Programs" des "US Departement of Defense":

- Robustheit: die Fähigkeit, aufgaben-, situations- und bedingungsübergreifend effektiv zu bleiben
- Belastbarkeit: die Fähigkeit, sich von Unglücksfällen, Schaden oder einer destabilisierenden Störung der Umgebung zu erholen oder sich darauf einzustellen
- Reaktionsfähigkeit: die Fähigkeit, auf eine Veränderung der Umgebung rechtzeitig zu reagieren;
- Flexibilität: die Fähigkeit, mehrere Lösungsmöglichkeiten einzusetzen und nahtlos von einer zur anderen überzugehen
- Innovationsfähigkeit: die Fähigkeit, neue Dinge zu tun und die Fähigkeit, alte Dinge auf eine neue Art und Weise zu tun
- Anpassungsfähigkeit: die Fähigkeit, Arbeitsprozesse zu ändern und die Fähigkeit, die Organisation zu ändern.

[Anmerkung R.K.: Es geht stets um "Fähigkeiten". Also um "weiche" Ziele. (Siehe **Agil**.)]

[vgl.: David S. Alberts, Richard E. Hayes, Wilfried Honekamp (Übersetzer): Power to the Edge. Re Di Roma-Verlag; 2009]
Diese Definition versteht unter "Flexibilität" im Gegensatz zu allen anderen Sichtweisen von "agil" nicht nur ein inkrementell-adaptives Vorgehen in kleinen Schritten ausgehend von einem "JEDUF", sondern lässt auch ein stark plangetriebenes, "BDUF"-basiertes Vorgehen dann zu, wenn es für die aktuelle Situation passender und effizienter ist.

[Anmerkung R.K.: Aus meiner Sicht sind
- Robustheit
- Belastbarkeit

- Reaktionsfähigkeit
- Flexibilität
- Innovationsfähigkeit
- Anpassungsfähigkeit

alle "weiche" Ziele (soft Skills). Sie sind nicht wirklich messbar; wünschenswert allemal, sind sie doch kaum quantifizierbar, schon gar nicht bei Menschen oder Teams: Fördern solche "weichen" Ziele das Wunschdenken? (Als Beispiel die Aussage vieler heutigen Teams: "wir sind agil, wir haben einen Stand-Up jeden Morgen"). Nicht lachen; es ist wirklich wahr!]

Fazit aus meiner Sicht

"Agil" scheint sehr unscharf definiert zu sein, es ist ein Modewort (alle Unternehmen wollen "agil" sein, für viele bedeutet das lediglich, täglich ein Stand-Up Meeting zu haben).
Der Begriff "agil" fördert das Wunschdenken und beansprucht für sich, eine Art "Generalschlüssel" für alle Probleme zu sein.

Über das Agile Manifest und die Prinzipien

Aus Hans-Peter Korn; Die "agile" Organisation: Vom Hype zum Daily Business. Diese vier Werte und zwölf Prinzipien [http://agilemanifesto.org/iso/de/principles.html] sind, dem Charakter eines Manifests entsprechend, als Appell zu verstehen und nicht als "verifizierte Arbeitsregeln". Sie dogmatisch als Handlungsanweisungen für die Software- oder Produktentwicklung in unterschiedlichsten Situationen zu betrachten wäre eine Fehlnutzung. [vgl: Andrea Janes, Giancarlo Succi: The Dark Side of Agile Software Development. Free University of Bolzano. Online in http://darkagilemanifesto.org/dark-side-of-agile-janes-succi-splash-2012.pdf]

Manifesto

Hier das Manifest, der Vollständigkeit wegen.
From: http://www.agilemanifesto.org

Manifesto for Agile Software Development
We are uncovering better ways of developing software by doing it and helping

others do it.
Through this work we have come to value:
1- Individuals and interactions over processes and tools
2- Working software over comprehensive documentation
3- Customer collaboration over contract negotiation
4- Responding to change over following a plan
That is, while there is value in the items on the right, we value the items on the left more.

Eine Übersetzung ist:
Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:
- Individuen und Interaktionen mehr als Prozesse und Werkzeuge
- Funktionierende Software mehr als umfassende Dokumentation
- Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
- Reagieren auf Veränderung mehr als das Befolgen eines Plans
Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.

Einige Gedanken zu den "Werten" des Manifests:

1- Also, "agil" sind Individuen ohne Tools die wilde Interaktionen ohne wesentliche Prozessorientierung zu tun pflegen. Verschwenderische Vorgehensweise.

2- Das Ergebnis wird dann oft: "Unmaintable SW and uncomprehensive documentation". Also für die Katz, die Entwicklung. Verschwenderische Arbeitsweise.

3- Echt krass; die Kunden werden geradezu ermuntert, ständig Änderungen und neue Funktionalitäten zu fördern! Verschwenderische Prinzipien.

4- Alles Andere als zielorientiert. Verschwendete Gedanken allesamt.

Aus meiner Sicht folgt daraus: Das "Manifest" ist noch nicht richtig anwendbar, ca. 95% müsste noch korrekt überarbeitet werden.

Die 12 Agile Prinzipien

From: http://www.agilemanifesto.org/principles.html

Principles behind the Agile Manifesto

We follow these principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity -the art of maximizing the amount of work not done- is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Eine Übersetzung ist:

Wir befolgen folgende Prinzipien:

1. Den Kunden durch die rechtzeitige und kontinuierliche Lieferung wertvoller Software zufrieden zu stellen ist unsere höchste Zielsetzung.

2. Wir begrüßen Änderungen der Anforderungen, auch wenn diese spät in der Entwicklung kommen. Agile Prozesse nutzen den Wandel für den

Wettbewerbsvorteil des Kunden.

3. Wir liefern funktionierende Software häufig, alle paar Wochen oder Monate, wobei wir die kürzeren Zeiträume bevorzugen.

4. Geschäftsleute und Entwickler müssen täglich im Projekt zusammen arbeiten.

5. Setze motivierte Individuen in den Projekten ein. Gib ihnen die Umgebung und Unterstützung, die sie brauchen, und vertraue ihnen den Job bestens zu erledigen.

6. Die Konversation von Angesicht zu Angesicht ist die effizienteste und effektivste Art der Informationsweitergabe an und in einem Entwicklungsteam.

7. Das primäre Maß für Fortschritt ist die funktionierende Software!

8. Agile Prozesse fordern nachhaltige Entwicklung. Die Sponsoren, Entwickler und Benutzer sollten ein gleichbleibendes Tempo ohne Unterbrechung einhalten.

9. Technische Exzellenz, gutes Design und deren fortwährende Beachtung verbessern die Agilität.

10. Einfachheit ist essentiell.

11. Die besten Architekturen, Anforderungen und Designs entstehen in Teams, die sich selbst organisieren.

12. Das Team reflektiert in regelmäßigen Intervallen darüber, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

Siehe auch: https://de.wikipedia.org/wiki/Agile_Softwareentwicklung

Einige Gedanken dazu:

Zu 1. "Höchste Zielsetzung", "Wertvolle SW": Pompös. Dieser Punkt ist als schwammiges Wunschdenken zu kennzeichnen. Ist das nicht Reklamesprache, um den "Kunden" einzulullen?

Zu 2. Es soll gelten: Änderungen zu minimieren. Diese "auch spät im Projekt willkommen zu heißen" bedeutet nichts anderes, als den Kunden überglücklich machen zu wollen. Solche Mehrarbeit zu begrüßen ist nicht nur unprofessionell sondern auch (höchst) propagandistisch.

Zu 3. Schwacher Punkt. Es ist wichtig, dass das Ergebnis funktioniert, und

zwar so gut wie möglich. Das die SW lange vor Ende des Projektes "arbeitet", und auch diese (so) oft (wie möglich) zu liefern; das kann kein wirkliches Ziel sein. Ziel ist das Endergebnis. Und darauf sollte fokussiert werden.

Zu 4. Die täglichen Meetings zeugen von Wunschdenken. Eine enge Zusammenarbeit ist erwünscht, doch "täglich" ist glatt übertrieben. Will man hier auch den Kunden mit dem Wort "täglich" propagandistisch angehen? Mit dem Wort "Busines People" fühlt sich der Kunde (Behörde, Endkunde) nicht direkt angesprochen; aus Sicht des Kunden ist die Leitung der Entwicklerfirma gemeint.

Zu 5. Gute Idee. Eine allgemeine gute Idee: soll sie fortan von den "agilen" gepachtet werden?

Zu 6. Aber halt nicht immer. Ausnahmen existieren. Ich halte erstaunlich wenig von solchen allgemeinen generalisierenden propagandistischen Aussagen. Die Punkte 5 bis 7 sind eher allgemeine gute Praktiken, wurden aber sicher nicht von den Autoren des "agilen Manifests" "erfunden".

Zu 7. "Funktionierende" SW als "Fortschritt" zu verkaufen halte ich für ziemlich gewagt. Sollen Komplexität, Anforderungen, Qualität, Risiken u.a. unwichtig sein, solange die Software "Funktionieren tut" ?

Zu 8. Nur solange die Entwickler bezahlt werden, werden sie auch entwickeln. Diese Ideen sind nicht ausschließlich "agilen Prinzipien", auch wenn das Wort vorangestellt wurde.

Zu 9. Scheinbar waren die Schreiber des Manifests ab Punkt 8 schon etwas müde. Schöne Wörter, zusammenhangslos aneinandergereiht. Propaganda halt.

Zu 10. Einfachheit ist schon (sehr) wichtig. Doch "Essential" ? Na ja. Außerdem ist "Einfachheit" ganz sicher nicht "die Kunst, die nicht verrichtete Arbeit zu maximieren". Im Gegenteil; oft muss ein Profi mehr Arbeit verrichten um ein System einfacher zu gestalten. Schwache Aussage.

Zu 11. Wunschdenken: Gute Architekturen werden von guten Architekten entwickelt. Gute Anforderungen werden von guten Requirement-Engineers geschrieben. Gutes Design wiederum von guten Designers erstellt. Diese Arbeiten haben mit einem Team eher wenig zu tun. Ziemlich egal wie sehr sich das Team "selbst organisiert". Möglicherweise ist der Irrtum von den sogenannten "selbstorganisierten Teams" hier entstanden. Siehe hierzu **selbstorganisierende Teams** unter "Mythen".

Zu 12. Gutes Prinzip. Sehr gut sogar (Richtung Kaizen). Eine Exklusivität von agilen Teams? Eher weniger. Nur gekonnt kopiert.

Auch hier (wie bei http://www.agilemanifesto.org) sollte man stark (zum Besseren) überarbeiten.
Vom ursprünglichen Manifest wird wahrscheinlich kaum etwas übrig bleiben.

Fazit aus meiner Sicht

Zum Nachdenken: Wie "agil" ist in Wirklichkeit "Agil"?

Mein Fazit zu dem Manifest:
Es handelt sich um pompöses Gequatsche: "erschließen bessere Wege", "anderen dabei helfen", "Werte zu schätzen gelernt".
Meiner Meinung nach haben sich einige netten Entwickler, die sonst kaum eine Ahnung von Projektarbeit hatten, zusammengetan (hatten wahrscheinlich gerade nichts Besseres zu tun) und eine Art Wunschlistenpamphlet geschrieben.
Darunter haben heute noch Tausende von Projektmitarbeiter zu leiden.

Im Grunde genommen besagt das Manifest folgendes: "Wir wollen

- Individuen und Interaktionen; kein Management, keine Analysten und keine Designer (tun wir selbst)
- Funktionierende Software; keine Dokumentation, wir sind nur der Freiheit des Geistes verpflichtet
- Zusammenarbeit mit dem Kunden; direkt ohne Management, ohne Verträge, ohne Verpflichtungen (wir liefern das, wozu wir gerade Lust verspüren)
- Reagieren auf Veränderung; kein Plan, tun und lassen was wir möchten".

Das mit den "Veränderungen" ist keine Lappalie, wie Sie weiter unter bei **Änderungen** lesen können. Jeder, der Programme geschrieben hat weißt, dass es einmal lauffähig noch Tage und Monate damit verbracht werden können, das Aussehen der Software (wirklich!) ad infinitem zu verfeinern. Der Punkt mit dem "auf Veränderungen zu reagieren" ist ein Fass ohne Boden; müsste der Giga-Schreck jedes Leiters sein, der sich ansatzweise mit der Projektarbeit auskennt.

Mein Fazit zu den 12 Prinzipien:
Inhaltslos: Punkte 1, 5 und 9.
Unflexibel: Punkte 4, 6 und 7.
Folgerung unlogisch: Punkte 8 und 9.
Falsch: Punkte 2, 10 und 11.
Meiner Meinung nach sind die Prinzipien schwammig, schwach und lahm. Eher

eine nicht besonders gekonnte Propagandaschrift. Aber gelungene, warum auch immer.

Über die sogenannten "agilen" Methoden

Meine Erfahrung bei den so genannten "agilen" Teams ist, dass es sich um ein Schlagwort handelt; spricht das Management an und darüber hinaus ist es ein Vorwand für aufgeblähte Prozesse.

Die "Diskussion" mit den Agilen Methoden scheinen für mich eher eine Glaubenssache zu sein. Die Fokussierung wird auf Form-Sachen gelegt, nicht auf Inhalt. Es sollten mehr oder minder "blind" bestimmte Muster angewandt, ohne dass Kritik und Verbesserungen in Betracht gezogen werden dürfen.

Meine Erfahrung mit den so genannten "agilen" Teams ist,
a) es handelt sich um ein Schlagwort (Buzzwort) für das Management, und
b) es ist ein Vorwand für mehr Verwaltung.

Methoden liefern nicht die von ihren Anhänger erwarteten Ergebnisse, weil ihre Erwartungen auf einem grundlegenden Missverständnis der Software-Entwicklung basieren: es ist nicht eine Form der Herstellung, wo das Ziel viele Repliken des Prototyps zu produzieren ist, es ist eine Design-Ähnliche Aufgabe, die mit der Erstellung des Prototyps auch endet. Der Erfolg hängt von Wissen und ein gutes Urteilsvermögen: Methoden können das gar nicht ersetzen.

Gut geschriebener Text ist sehr oft besser als "Face-to-Face"-Kommunikation. Man kann später ein gut geschriebenen Text referenzieren, anstatt sich auf dem Gedächtnis zu verlassen. Man erzeugt immer gut geschriebenen Text, denn man muss sorgfältig denken. Auch technisch gesehen, ist es nicht möglich guten Code zu produzieren, wenn man nicht gut geschriebenen Text schreiben kann.

[Anmerkung R.K.: Das "Nicht-Schreiben" steht auch im Einklang mit dem allgemeinen "Sprintgedränge"; hetzen um schnell etwas fertig zu kriegen, kurzerhand etwas Code kopieren-und-einfügen (anstatt das Geschrieben anzupassen um nicht doppeltes Code zu haben), schnell eine mündliche Absprache (denn alle wissen es, keiner spricht das aber aus: "im nächsten oder übernächsten Sprint wird sowieso alles anders aussehen"). Daher macht geschriebene Kommunikation kaum Sinn im "Gedränge". (Siehe **Gedränge**).]

Zurzeit werde ich in Projekten eingespannt, wo die Mitarbeiter mit so viel "Prozess" und "Best Practices" belastet werden, das fast nichts von wirklichem Wert (Nutzen) erstellt wird.

Einfache User Stories können unmöglich die Komplexität einer SW auffangen. Damit läuft man in Teufels Küche hinein und nie wieder heraus.

[Anmerkung R.K.: Andererseits soll man die wichtige Rolle, die "User Stories" spielen, nicht verkennen: Wie soll man sonst einem "Dr. Phil." (oder Ähnlichem) eine technische Anforderung begreiflich machen?]

Flexible Entwicklungsmethoden; ohne vorab detaillierte Spezifikation kann man auch kein detailliertes Design erstellen. Das ändert die Art und Weise wie man Software baut: Refactoring wird regelmäßiger und Sie benötigen automatisierte Tests, um dass zu unterstützen.

[Anmerkung R.K.: Refactoring ist doppelte Arbeit und doppelte Arbeit ist Verschwendung. Erinnern Sie sich vielleicht an die "O-Notation" (wenn Sie etwas Informatik gelernt haben) um die Komplexität von Algorithmen zu beschreiben? Demnächst werden wir wieder mit so einer 'Notation' konfrontiert werden, nämlich um die Komplexität vom "Refactoring" als Funktion der Sprintnummer zu beschreiben. (Siehe dazu auch **O-Notation**).]

Refactoring ist Verschwendung. Bei einem BUFD hätte man einen großen Batzen Refactoring am Ende, bei den Agilen immer und immer wieder (bei jedem Sprint, bei jeder Iteration). Die Gretchenfrage ist: Welches verbraucht mehr Ressourcen?

In einer wirklich agilen Welt, nichts ist heilig. Man tut, was am meisten Sinn in einem gegebenen Kontext macht, um das Ziel zu erreichen.

Dass sich Menschen und Unternehmen einer Methode anpassen müssen und nicht umgekehrt zeigt mir, dass hier etwas mit der Methode selbst faul ist: Das erinnert mich eher an totalitären Systemen und an fundamentalistischen Religionen.

Die SW-Entwicklung mutierte zu einem Spiel.

Prinzipien die helfen praktische Dinge zu tun sollten falsifizierbar sein. Dies stellt ein Problem dar mit vielen Methoden, nicht nur mit den agilen Methoden. Agile Methoden verlangen einen "Backlog" von Funktionalitäten und priorisieren diese. Sodann bearbeitet man den Backlog so weit es geht im Laufe des Sprints. Nun, die agilen Methoden "funktionieren" immer, **weil sie so definiert sind**. Also sind sie nicht falsifizierbar...

[Anmerkung R.K.: Ein Prinzip zu falsifizieren ist sicherlich sinnvoll, eine Methode zu falsifizieren jedoch nicht: So lange sie funktioniert soll es einem Recht sein. Denke ich. Das Problem liegt jedoch woanders...]

Inzwischen geschieht die Kommunikation, die in einem Standup vorkommen sollte, nach meiner Erfahrung selten dort. Und ich denke nicht, dass das ein Problem ist (*).
Ich denke, die Notwendigkeit der Standups ist ein Symptom für unzureichende Kommunikation. Standups können eine Lösung darstellen, aber sie sind alles andere als optimal.

[Anmerkung R.K.: Zu (*); es ist doch ein Problem: Verschwendung.]

Die einzige "Methode" die ich in der Wirklichkeit durchgehend arbeiten gesehen habe ist die mit dem mythischen Mann-Monat. Der Autor verwendet ein OP-Team als Modell: Eine einzige technische Führung mit voller Autorität, Entscheidungen zu treffen, in einer unterliegenden Schicht befinden sich die Assistenten samt dazugehörigen technischen Personal, und dann eine nicht-technische Person als Unterstützung. Die Leistung des Teams wird durch die Führung des Teams bestimmt. Die Größe des Teams und die Komplexität des Projektes wird maßgeblich durch den Leiter und ihre Fähigkeit, das Problem zu verstehen und Aufgaben zuzuweisen, bestimmt. (Siehe hierzu **OP-Team**).

[Anmerkung R.K.: Hier greift der Gedanke zu kurz, man macht es sich zu einfach: Ein OP Team ist ein sehr dezidiertes Team; hat ein klares Ziel, eine eindeutige Aufgabe. Eine SW Entwicklung ist im Allgemeinen jedoch erheblich komplexer.]

Doch meine Erfahrung aus der Praxis ist, dass schlechte Manager in der SW-Entwicklung einige vom "Agile" abgeleiteten Mechanismen nutzen um ein völlig kontraproduktiven Mikromanagement zu implementieren.

So zum Beispiel:

Ein Manager hat wohl keine Zeit, Code zu lesen und die technischen Fragen im Detail zu verstehen. Aber sie können die Kanban-Tafel sehen. Also denkt der Manager, dass die Kanbantafel die Realität ist.

Die Kanbantafel ist ein stark vereinfachtes und oft ziemlich falsches Modell der Wirklichkeit. Die Wirklichkeit ist das Programm, die Entwickler und die Bugs... und die Bugs sind "wirklicher".

Das Kanban, die "Story Points", die Sprints, alle tragen dazu bei, ein falsches Bild der SW-Entwicklung zu fabrizieren, dergestalt, dass die SW-Entwicklung wie das Mauern verstanden wird, d.h., eine in hohem Maße vorhersehbare Aufgabe, die mit konstanter Geschwindigkeit durch mäßig kompetente Mitarbeiter getan werden kann, mit wohlgeformten Batzen Arbeit (Stories), die man leicht verteilen kann ohne auf Abhängigkeiten zu achten. Solange die Wände alle die gleiche Höhe haben und die Ecken passen, ist alles OK.

Bitte merken: Software zu entwickeln ist nicht wie mauern!

Nun aber denkt der Manager, dass die Kanbantafel und die kurze Mitteilungen in den Standup-Meetings die Realität sind. Der Manager wird ziemlich frustriert, wenn es sich herausstellt, dass die Maurer die Ziegelsteine gerade nicht in einer regelmäßigen und vorhersagbaren Weise bearbeiten und die Aufgaben aus jedem Sprint nicht fertig werden. Das Projekt hinkt hinter dem Zeitplan.
Das Projekt ist also hinter dem Zeitplan; was geschieht nun im nächsten Sprint? Anstatt mehr Ressourcen zu finden (was schwer ist), oder Funktionen zu streichen (was in einer agilen Methode die richtige Antwort wäre), entscheidet der Manager
a) die Schätzungen für neue Aufgaben runter zu diskutieren (was das Kanban weiter weg von der Realität bewegt);
b) die Entwickler weiter unter Druck setzen.
(Mehr zu Kanban weiter unten, siehe **Kanban**.)

Ein Grundprinzip der "Agile" ist das Lösen von Problemen in "Face-to-Face"-Meetings. Doch im Laufe der letzten zehn Jahre haben Erbsenzähler-Manager Offshore-Entwicklung mit Teams in Indien, China usw. organisiert, weil die Kosten pro Stunde von Maurern (ich meine, Software-Entwickler) dort viel billiger ist. Jedoch a) stehen diese Teams für eine "Face-to-Face"-Meetings nicht zur Verfügung; b) oft sind sie in Zeitzonen wo der Arbeitstag nicht einmal überlappt; c) Sprache/Akzent und kulturelle Unterschiede können ein Hindernis für eine effektive Kommunikation sein; d) diese Teams sind meist weniger erfahrene Maurer (Verzeihung, Software-Entwickler). Natürlich lebt auch in diesem Fall der Manager weiterhin in dem Glaube, dass das Kanban die Wirklichkeit entspricht. Er mag das auch sehr, weil die Kanbantafel mit den Offshore-Teams (sehr) einfach zu teilen ist.
Nach einem Jahr oder so, verlassen die besseren Entwickler das Team und finden etwas weniger frustrierend zu tun. Bald ist die ganze Organisation in einer Todesspirale.

[Anmerkung R.K.: Das wirkliche Problem ist, dass die "agilen" Methoden wenig (sehr wenig) Substanz haben und auf Mechanismen setzen müssen, die nichts

mit "Agile" zu tun haben (wie Kanban z.B.; hat mit "Lean" zu tun, nicht aber mit "Agile"). Ein "Agilist" würde vielleicht an dieser Stelle behaupten, dass "Agile" nicht richtig gemacht worden ist. Ich hingegen behaupte, die "Agilisten" haben Kanban verhunzt.]

Ich habe überlegt, dass Software-Entwicklung viel mehr wie die Patientenversorgung in einem Krankenhaus organisiert werden soll, als wie bei Mauerarbeiten. Der Normalzustand der Software ist tief ungesund, mit vielen Bugs und Performance-Probleme. Das primäre Ziel des Entwicklungsteams ist es, dem Patienten am Leben zu halten: das heißt, die Fehler unter ausreichende Kontrolle zu halten, damit die Software für die Kunden nutzbar sein kann (ohne Abstürze, ohne ernsthafte Fehler und schnell genug, dass der Kunde sie gebrauchen kann). Jedes einzelne Problem ist ein Symptom für das wir einen erheblichen diagnostischen Aufwand (Debuggen) betreiben müssen um die zugrunde liegende Ursache zu verstehen, und dann sollte man mit verschiedenen therapeutischen Maßnahmen (Fehlerbehebung) versuchen, es zu lösen. Dazu kommt, dass wir oft gar nicht wissen, was funktioniert bis wir es ausprobiert haben.

[Anmerkung R.K.: Abgesehen davon, dass es ein schönes Gleichnis ist, pflege ich eine andere Sicht der Sache: Wenn der Patient "tief ungesund" ist, so haben wir es mit miserablen Design und grottenschlechten Qualitätssicherung zu tun, und wenn man so viel Aufwand für die Untersuchung der Symptome betreiben muss, so deutet das eher auf einer "agilen" Entwicklung, wo die Entropie der Software regelmäßig außer Kontrolle gerät.]

Entwickler, wie viele andere Berufe, mögen es gar nicht die zu verrichtenden Arbeit in beliebigen Messeinheiten zu quantifizieren, damit fühlen wir uns wie Idioten behandelt. Jedwede Schätzung ist so oder so unschön; sowohl wenn wir die Zeit unterschätzen als auch wenn wir sie überschätzen, so oder so sind wir die gelackmeierten.

[Anmerkung R.K.: Es sind doch gerade die "Agilisten" die postulieren, dass erst "sich selbst organisierende Teams" in der Lage wären, mit Komplexen Systemen (als Gegengewicht zu den "Komplizierten" Systemen) umzugehen und dass wiederum agile Methoden, wegen der "Agilität", als Methode erst mit komplexen Systemen umgehen könnten. Und dann das. Einfach die Arbeit in Batzen zerstückeln und in Time-Boxen (Zeitspannen) rein pressen. Genial einfach, einfach... mir fehlen die Wörter.]

Analysten kommen nicht mehr vor und Projektmanager wurden auf "Teamleiter" und "Scrum Master" degradiert. Leitungsaufgaben sind weggefallen und sie haben nicht wirklich irgendetwas unter Kontrolle, außer Rituale die das Team-Konsens diktiert [N.B.: Nicht mal das.].

Agile Methoden bringen schlechte und mittelmäßige Entwickler dazu, mittelmäßig zu entwickeln, jedoch auch die guten Entwickler fangen an, mittelmäßig zu entwickeln. Einmal im Sumpf der Mittelmäßigkeit, gibt es kein Entrinnen mehr. Die guten bleiben mittelmäßig, die Mittelmäßige werden nie gut und die Schlechten fallen nicht mehr auf.

Es ist auch gewissermaßen neurotisch: Wenn man bemängelt, dass die "agilen" Methoden nicht agil, rigide und unflexibel sind, dann erklären die, die dafür sind, dass man sie anpassen soll. Sollte man sie aber angepasst anwenden und sie immer noch nicht funktionieren, so beteuern die, die dafür sind, dass man sie "falsch" einsetzt und dass man sie genau so zu implementieren hat, wie sie definiert wurden.

Agil ist nicht immer falsch, manche (mittelmäßigen bis schlechten) Teams werden sicherlich mit dem Schlimmsten z.B., Scrum sogar, gut fahren.

Vielleicht sind die "agilen" Methoden so weit gekommen, weil viele nicht "MINT" Absolventen hier, aufgrund des vermehrten Kommunikationsschwer-punktes, eine Arbeits- bzw. Einsatzmöglichkeit witterten?

Hier steht wie man agil falsch machen kann:

Organisationen sparen sich sorgfältige Planung und schieben alle Unsicherheiten zu den Entwickler-Teams zu, hoffen dabei, dass die Entwicklung Iterationen verkürzen kann und die tägliche Sync-Ups auf magischer Weise alles Übel heilen und die Dinge richtig machen kann. Software-Entwicklung ist ein unglaublich "front-loaded" Prozess, das heißt, eine Menge Gedankenarbeit soll im Vorfeld investiert werden um zu wissen was und wie gemacht werden soll, bevor man es tut. Die Anwendung von agilen Prozessen kann das nicht ändern. Agil macht es einfach leichter, sich an Veränderungen anzupassen. Wenn eine durchdachte Planung und Gestaltung gespart wird, kann keine "Methode" die SW retten. Am Ende aber sind die Entwickler Teams immer schuld, weil sie angeblich das Endprodukt hätten liefern sollen.

Das Management sah in den agilen Entwicklungsmethoden eine Möglichkeit "in den Tag hinein" zu managen und dabei die Entwickler maximal auszunutzen. In erster Linie, sollte eine agile Entwicklung die Entwickler in der Lage versetzen, bessere Software zu liefern, während sie proaktiv arbeiten und nicht schlichtweg unter Stress gesetzt werden. Die „Agilen“ sind kein Tool des Managements um die Entwickler-Teams täglich unter dem Mikroskop zu setzen

und zur Höchstleistung zu treiben. Agile wird am besten organisch durch starke Führung innerhalb der Teams durchgeführt, nicht durch einige Top-Down-Richtlinien oder durch Spritzen von IT-Berater.

Man achtet nur auf das "Wie", ohne "Warum" zu fragen. Es ist einfach, einige oberflächlichen Ritualen zu implementieren und sich "Agil" zu deklarieren. Aber mit einem täglichen Sync-Up macht man sich nicht agil. Es ist nur eine Zeitverschwendung, wenn die Entwickler von diesen Treffen nichts haben. Eine kürzere Iteration zu haben macht die Entwicklung nicht auf magischer Weise schneller oder mit weniger Fehlern. Es kann aber ein mehr an Verwaltungsaufwand durch die häufigeren Unterbrechungen bedeuten. Agile soll eigentlich helfen, sich auf Veränderungen anzupassen. Dies gilt für das Entwicklungsprozess auch. Die Teams sollten einen holistischen Ansatz nehmen und verschiedene Dinge ausprobieren, bis sie "Sweet-Spots" finden.

Wenn man die überwältigende Menge an nicht sehr erfolgreichen "agilen" Projekten, oder gar die spektakulär gescheiterten Agile Projekte betrachtet, wird man immer einen gemeinsamen Refrain hören: "Sie haben Agile falsch gemacht" Die meisten Organisationen kriegen zumindest ein paar agilen Eigenschaften nicht auf die Reihe. Inzwischen fühlen sich die Entwickler krank vom Scheitern und die ganze Gesellschaft fühlt sich enttäuscht. Bald verwerfen sie die Methode oder sie passen sie zu etwas, was mittelmäßig funktioniert und nennen es immer noch "Agile."

Früher gab es eine Fernseh-Serie in Großbritannien die "Red Dwarf" hieß. Ich bin kein großer Fan von der, aber es gibt eine Episode, welche die Agile Welt wirklich darstellt. Es ist die Episode, in der eines der Besatzungsmitglieder des Raumschiffes verrückt wird; er trägt karierte Klamotten und führt bizarre Strafen ein, wie z.B. Sauerstoffentzug.
Dies ist, was Agile für mich ist. Das Problem ist, dass Agile durch völlig verrückten Ideologen übernommen ist, so dass die Entwicklung sich eher wie eine Strafe als wie Arbeit anfühlt.

1. Agile kann u.U. bei der Web-Entwicklung funktionieren, sobald der Kunde etwas will, wird geliefert.
2. Als Entwickler habe ich keinen Anspruch mehr zu arbeiten oder besser zu werden, einfach nur die Story Points abarbeiten. Kein Refactoring. Kein robustes System.

Wie viele Programmierer, war mein natürlicher Zustand ruhig vor einem Computer zu arbeiten. "Face to Face"-Meetings, Code-Reviews, Paar-Programmierung, die Arbeit den Stakeholder demonstrieren, sowie alle diese alberne Übungen, die viele Agile Teams übernehmen, sind das genaue Gegenteil davon, wie die meisten Programmierer zu arbeiten pflegen.

[Anmerkung R.K.: Und, meiner Meinung nach, wie gute SW entsteht.]

"Velocity": Was zum Teufel ist das? Es ist viel schlimmer als LOC. Manche MBA-Typen sind begeistert von der Velocity: Es ist eine quantifizierbare Zahl die sie in Excel eintragen können. Sie wissen nicht, was sie bedeutet. Aber es ist eine Zahl!

[Anmerkung R.K.: Velocity (nicht Geschwindigkeit, wohlgemerkt) ist ein Schlagwort. Typisch für die "Agilen", werden Begriffe neu belegt und propagandistisch verkauft. Das 'Management' ist begeistert. Siehe **Velocity**.]

Für alle die gerne beantworten "Du benutzt agile falsch: das ist nicht Agile": Wenn 90% der Praktiker $FOO nicht korrekt zum Arbeiten kriegen können, dann liegt das Problem nicht an den Anwendern, sondern $FOO selbst ist das Problem. "Das" weiß ich.

Als Ingenieur möchte ich eine sinnvolle Arbeit. Nichts sinnvoll kann in zwei Wochen abgeschlossen sein. Agile ist nichts anderes als eine Beleidigung, eine Infantilisierung von Ingenieurarbeit. Planning Poker? Post-its? "Gedränge"? Das ist ein schlechter Witz.

Die Entwickler (vor allem Sr.) fühlen sich nicht respektiert bzw. nicht vertraut, wenn sie angehalten werden zu erzählen: "Was hast Du gestern getan, und was tust Du heute?". Dies ist eine sehr erniedrigende Frage; und diese sollte man nur einem Mitarbeiter ganz unten im Unternehmen fragen dürfen.

[Anmerkung R.K.: Ein Mitarbeiter fasst das als Mikromanagement auf. Ein Ingenieur erkennt die Zeitverschwendung. Ein Entwickler sucht einen Vorteil, vergebens. Ein Tester versteht den Sinn nicht. Das 'Management' versteht kein Wort aber fühlt sich "bestätigt".]

Mikromanagement im Allgemeinen. Ich habe nur X Stunden pro Woche die ich allozieren kann, und schon allein 40% geht mit diesen BS-Meetings zum Suchen von "Hindernisse" und "Blocker" flöten, wenn der größte Blocker gerade diese unnütze Sitzungen, ständige "Pings" und Telefonanrufe und Drop-bys. [...! Anmerkung R.K.: nicht zu übersetzen].

Was "Agile" zunehmend vergiftet hat sind die "Überflüssigen", die uns über die Jahre überladene Tätigkeiten wie User-Story-Abschätzen, Sitzungen zum

Poker-Planen halten, übertriebenes Tracking von Software, Burn-Down-Charts zeichnen, und 30-Personen-Stand-Ups für ein 4 Personen Entwicklerteam (Beispiel aus der Praxis!), die wenig oder gar nichts mit dem Entwickeln einer Software zu tun haben, eingebrockt haben.

Für mich fühlt sich das irgendwie erniedrigend: Als Software-Entwickler kann man sich oft selbst etwas beweihräuchern und so tun, als ob man fast ein Akademiker ist; man arbeitet an etwas Intellektuelles während man Kaffee trinkt und angenehme Gespräche führt.
Sobald man aber jeden Tag ein Stand-up Meeting hatte, ist die Vorstellung fort und man fühlt sich eher wie ein Fabrikarbeiter.

Noch ein Problem mit Agile: es ist nicht geeignet für ein Brocken-Problem, das nicht leicht gelöst werden kann: Man kann nicht jeden Abgrund mit zwei kleinen Sprüngen überqueren, manchmal ist ein großer Sprung nötig.

[Anmerkung R.K.: Das ist halt das Thema mit der "Inflexibilität" der ach so "flexiblen" Agilen.]

"Agile" verfällt unweigerlich in einer Geschwindigkeitsmessung (*) durch das Management. Ich habe noch nie einen agilen Apologet, der, mit dem Scheitern von agilen Projekten konfrontiert, nicht gemeint hätte, dass es "nicht wirklich agil" war. Auch dieses Thread hier ist voll mit solchen Aussagen. Wenn sie recht hätten, dann würde es sich zeigen, dass diese Methoden eine hohe Neigung zum Scheitern inne haben. Und wenn sie falsch liegen und die Entwickler recht behalten, so stellen die negativen Bewertungen einer solchen Last, dass sich die Frage stellt, was für eine Verbesserung diese Methoden darstellen sollen.

[Anmerkung R.K.: (*) Achtung, Velocity ist hier gemeint, nicht "Geschwindigkeit". Diese Schlagwort-Velocity ist schwammig definiert ("The points completed per Sprint is called the velocity of the team"; Die 'Velocity' sind die Punkte, die das Team im Sprint abgearbeitet hat) und willkürlich messbar. Siehe **Velocity**.]

Kein Fazit

Nein, ich habe an dieser Stelle nichts hinzuzufügen.

Warum agile Projekte scheitern

Hier sind meine 4 wichtigsten Gründe für agile Projektfehler:

Man schafft nicht die Hype-Kurve zu folgen: Scrum (und "Agile" im Allgemeinen) wird als die Lösung zu allen Probleme der traditionellen Projekte angepriesen. In der Tat sind sie es nicht, Agile geht nur mit diesen Problemen auf einer anderen Weise um. Die Vorgehensweise von Scrum/Agile macht Probleme nur etwas früher im Laufe des Projektes offensichtlich. Ursache für das Scheitern von den "Agilen" ist die Unfähigkeit, diesen Problemen zu adressieren.

[Anmerkung R.K.: So wie ich es sehe werden die Probleme früher sichtbar, das gestattet jedoch die Probleme zu "ignorieren" so lange sie nicht "dringend" werden (man muss also unterscheiden können zwischen "wichtig" und "dringend"). Man tut was "wichtig" ist (bzw. als solches gilt), bis die Probleme dringlich werden. An diesem Punkt kann es schon zu spät sein. Bei gute Prozesse müssten Mechanismen existieren, welche die Abarbeitung von Problemen erzwingt, wobei es Schnuppe ist, welche Entwicklungsmethode Anwendung findet.]

Fehler richtig zu definieren und Anforderungen zu priorisieren. Scrum braucht die "Product Owner" Rolle um dies zu bewerkstelligen. Das ist eine unglaublich schwere Rolle, die eine Kombination von Business-Analyse-Fähigkeiten, technisches Wissen, Kommunikationsgeschick und Chuzpe abfordert. Diese Fähigkeiten sind sehr selten in einer Person zu finden. Ohne diese Rolle richtig besetzt oder irgendwie dafür zu sorgen, dass anderweitig die Skills besetzt sind (vielleicht, indem sichergestellt wird, dass im Team Analyse-Fähigkeiten vorhanden sind) haben agile Projekte ein sehr hohes Risiko des Scheiterns.

Mangelndes Verständnis, dass Agile/Scrum mehr Struktur und Disziplin als herkömmliche Projekte erfordert. Eine überraschend große Zahl von Mitarbeitern denken, dass agil bedeutet, dass sie nicht planen müssen, keine Dokumentation schreiben müssen und dass sie erst dann reagieren, wenn der Kunde seine Meinung ändert, denn das ist agil. Richtig? Die Realität könnte von der Wahrheit nicht weiter entfernt sein. Als Agiles Projektmanagement / Scrum Master gibt es zwei Dinge, bei der Sie nie Kompromisse machen sollten:
- Iteration / Sprint Enddatum. Dieser Stein im Boden ist der Schlüssel zur Fähigkeit, das Projekt zu überwachen und zu steuern
- Definition für Fertig (definition of done). Am Ende des Sprints ist erst dann etwas fertig, wenn es ein lieferbares Etwas ist. Alles, was noch nicht fertig ist, ist halt nicht fertig. Ein 90% fertig gibt es nicht.

[Anmerkung R.K.: Wenn Agile mehr "Struktur und Disziplin" als "herkömmliche Projekte" erfordert... eine dumme Frage zwischendurch: Sind "herkömmliche Projekte" jetzt "Wasserfall" Projekte?
Und noch eine klitzekleine dumme Frage: Wenn "Wasserfall" auch mehr

"Struktur und Disziplin" hätte... Würde "Wasserfall" vielleicht auch nicht (besser) schneiden als "Agile"?
Und soll also "Agile" mehr "Struktur" haben als "Wasserfall"?
Warum fühle ich mich gerade jetzt etwas verwirrt?
Nun, "Wasserfall" ist, wie weiter oben beschrieben, ein "Buhmann".
Folglich:
Gehe ich also richtig in der Annahme, dass "Struktur und Disziplin" bei "Agile" tendenziell bei einem absoluten Maximum der (Optimierungs-)Kurve liegen soll?
Wenn dem so ist: Wo liegen überhaupt die echten Vorteile von "Agile"?]

Risikotransfer. In der traditionellen Projekten, kommt der Kunde zu Ihnen mit Anforderungen und Sie antworten mit einem Zeitplan und ein Budget. Sobald diese vereinbart sind, haben Sie die Verantwortung übernommen, das Projekt zu liefern. Alle Fehler sind Ihre Fehler (ich spreche eher in einem psychologischen als in einem Vertraglichen Sinn hier). Scrum/Agile erkennt die Unsicherheit, die bei der Bereitstellung von Projekten vorhanden ist, und fordert vom Kunden einen Anteil an dieser Verantwortung zu übernehmen durch die Akzeptanz, dass einige ihrer Anforderungen nicht geliefert werden können. Einige Kunden sind nicht bereit bzw. nicht in der Lage (wegen der Kultur ihrer Organisation), diese Verantwortung zu übernehmen. Wenn dies der Fall ist, dann hat der agile Projekt auch hier ein sehr hohes Risiko des Scheiterns.

[Anmerkung R.K.: Ich frage mich, ob der Autor vielleicht aus der Psychoszene kommt: Irgendwie passt dazu, dass der Kunde "Verantwortung" für das Projekt, das der Auftragsnehmer gerade abarbeitet, übernehmen soll. Der Auftragnehmer übernimmt also nur die Verantwortung, die ihm genehm ist, die andere übergibt er (feierlich?) dem Auftraggeber zurück.
Bei einem ersten Blick kann man sich solche Kunden nur wünschen. Aus Erfahrung jedoch: Finger weg! In diesem Falle liegt das hohe Risiko des Scheiterns daran, dass der Kunde unkundig ist und somit in die Zahlungsunfähigkeit (oder Schlimmeres) rutschen kann.]

Auf den Punkt gebracht macht Agile viele Annahmen, die in den meisten echten Szenarien unrealistisch sind:

- jeder Ingenieur in einem Team kann kann jede Aufgabe mit gleicher Geschwindigkeit und Qualität als jeder andere bearbeiten
- Backlog Stories können nahezu unbegrenzt geteilt werden, so dass sie in den Sprints passen
- das Produkt soll am Ende eines jeden Sprints in einem lieferbaren Zustand sein
- ein Scrum-Daily jeden Tag ist ein sinnvoller Weg den Fortschritt im Sprint zu verfolgen.

[Anmerkung R.K.: Viele werfen "Agile" und "Scrum" in einem Topf. Wobei ich "Scrum" für weit schlimmer als nur "Agile" halte.]

Ein paar Dinge, die "Agile" nicht zu adressieren schafft, aus meiner Sicht:

- verschiedene Ingenieure haben unterschiedliches Wissen und Können, es ist also absurd abstrakte Story Points den Backlog Stories zuzuweisen ohne das zu berücksichtigen
- Backlog Stories in einer Vielzahl von kleineren Teilen zu brechen, um in einen Sprint zu passen, bringt zusätzliche Komplexität
- 90% der (komplexen) Software kann nicht alle zwei Wochen in einem lieferbaren Zustand sein
- Ingenieure hassen Treffen, die nicht unbedingt notwendig sind
- F&E und Design sind maßgebend in jedem Projekt.

[Anmerkung R.K.: Zusätzliche Komplexität bedeutet zusätzliche Verschwendung.]

[Anmerkung R.K.: Absolventen der Erziehungswissenschaften andererseits finden Meetings ganz wichtig.]

Der größte Teil des Berichts bewertet Kultur und Denkweise als wichtigster Grund für das Scheitern von Agile.
Wenn man nicht weiß warum etwas passiert, kann man immer die Schuld auf die "Kultur" geben. Es ist wie mit den Wissenschaften; wenn Jemand nicht weißt warum etwas passiert, dann kann man immer noch meinen, der „Gott" hätte es getan.

[Anmerkung R.K.: Bis jetzt ist die 2. Fraktion der "Agile" Apologeten nicht aufgetaucht; jetzt aber. Es gibt 2 Fraktionen von "Agile" Apologeten; die ersten (lautesten) erklären das Scheitern von den "agilen" Methoden mit der falschen Anwendung der Methoden. Die 2. Fraktion erklärt das Scheitern damit, dass die Mitarbeiter die falsche "Kultur" oder die falsche "Denkweise" oder ähnliches haben oder hätten.]

Die künstliche Trennung in sehr kleinen Teilen auf der Seite der Schnittstelle bedingt auf der technischen Seite einen Verlust des Überblicks und erzeugt Ineffizienz bezüglich die Wiederverwendung von Code; für den gleichen Zweck wird oft Code mehrmals geschrieben. Nicht alle Systeme können in Stückchen gebrochen werden, und jemand muss den Überblick behalten; das war die Rolle des Analytikers in den nicht agilen Umgebungen.

[Anmerkung R.K.: Das Nicht-Verwenden von Code ist eine Folge von dem Nicht-Vorhanden-Sein einer Planung und bedingt durch die Hetze bei den Sprints. (Siehe **Gedränge**).]

Über XP

Es existiert zwar nicht "die eine verbindliche" Definition von XP; alle seine Beschreibungen beruhen jedoch auf diesen fünf "Werten":

- Kommunikation
- Einfachheit
- Rückmeldung
- Mut
- Respekt.

[Anmerkung R.K.: Wie bei dem Thema "Agil" gesehen, es sind alle weiche bis sehr weiche Begriffe. Vor Allem "Mut" und "Respekt" klingen so schön nach "guck her! Wir sind die Guten!"]

XP beschreibt Spielregeln und Techniken, wie 6 bis 10 Entwickler als Team in enger Zusammenarbeit mit dem Benutzer schrittweise funktionierende Software mit höher Qualität realisieren können.
XP beschreibt keine Spielregeln und Techniken zur koordinierten Arbeit mehrere Teams an einem umfangreichen Softwaresystem. Für derartige Situationen müssen zusätzliche Spielregeln und Techniken des Projektmanagements (z.B. PRINCE2 oder DSDM) bzw. des Programm- und Portfoliomanagements (z.B. SAFe) angewendet werden. Essentiell ist, dass diese zusätzlichen Spielregeln und Techniken vereinbar sind mit:

- den fünf Werten des XP
- der Realisierung überprüfbarer Softwareinkremente in kurzen Iterationen von maximal 4 Wochen
- einem eher groben "JEDUF"
- der kontinuierlichen Konkretisierung und Adaption der Benutzeranforderungen im Verlauf der SW-Entwicklung primär im persönlichen Dialog
- der kontinuierlichen und raschen Verfügbarkeit der Benutzer bzw. Kunden (bzw. deren entscheidungsbefugten Repräsentanten) für die Entwickler.

[Anmerkung R.K.: Hier auch die pompöse Sprache; "in enger Zusammenarbeit mit dem Benutzer" (besser sicher als 'in Zwietracht mit dem Management'), "funktionierende Software mit höher Qualität" (immerhin besser als 'nicht funktionierende SW mit höchster Qualität').]

Pair Programming: Praktisch, ja, einige Wochen vielleicht; wenn ein Anfänger hinter einem "Senior" sitzt. Sonst liegt die Produktivität bei ca. 10 bis 20%

mehr von dem, was ein Entwickler alleine schafft: Also ein Wirkungsgrad von max. 60%.

Pair Programming ist einfach zu viel des Guten für gut 80% der Programmieraufgaben, für die restlichen 20% ist es gerade gut genug. Brauchen Sie wirklich einen Beifahrer im Auto um in die Garage zu parken? Mit anderen Wörter; nicht jede Programmieraufgabe ist so gefährlich wie Gehirnchirurgie oder einen Linienflugzeug zu fliegen.

Pair Programming: die reinste Zeitverschwendung. Pair-Programming hat keine Vorteile, macht eigentlich nur dann Sinn, wenn die Mitarbeiter so wenig Ahnung haben bzw. so schlecht geschult sind, dass man besser mehrere Ahnungslose an eine Aufgabe arbeiten lässt um die Risiken zu reduzieren.

Und noch ein Punkt: Pair Programming ist einer von mehreren falschen Ansätzen; durch die Anwesenheit von 2 Programmierern wird versucht, das fehlende Design und die fehlende Systemanalyse zu ersetzen. Das ist kurzsichtig, nicht zu Ende gedacht; so als würde man einen Meisterkoch durch 2 Kellner ersetzen wollen. (Siehe **pair**).

"XP" ist aus vielen Gründen umstritten, doch eines der wichtigsten Problemen mit "XP" ist es, dass es evolutionäres Design anstatt geplantes Design postuliert. Wie wir erfahren konnten, kann evolutionäres Design aus 2 Gründen unmöglich funktionieren:
1. Ad-hoc Design-Entscheidungen und
2. Software-Entropie.

Fazit aus meiner Sicht

XP scheint ein erster Versuch von der Entwicklerseite gewesen zu sein, um sich von dem Management und der Zwischenschicht (Analytiker und Designer) zu distanzieren. Notgedrungen mussten sie also die Aufgaben dieser Zwischenschicht übernehmen.
Dementsprechend laienhaft ist dieser erste Versuch einer Definition von "agilen" Methoden auch aufgefallen. Erstaunlich weit sind sie aber mit dem Stuss gekommen. (Siehe **Agil**.)

Immerhin müssen sie einen Grund gehabt haben, um dies zu unternehmen. Sieht so aus, als wäre die Zwischenschicht "damals" sehr schlecht gewesen, und heutzutage ist sie so gut wie nicht vorhanden. Die Entwickler haben diese

Arbeit übernommen: Keine optimale Lösung, aber eine die billig ist und trotz Allem Ergebnisse liefert.

Über Scrum

"Scrum" ist bei Weitem die Methode mit den meisten Kritikpunkten, gleichzeitig jedoch die mit der weitesten Verbreitung. Das mutet befremdlich an: Wieso ist es so?

Eine Methode, die offensichtlich die Minderwertigste ist, ist aber dennoch die am meist Gefragte: Wie passt das zusammen?

Was läuft schief in den Köpfen des ‘Managements‘ oder, noch schlimmer, in den Köpfen des ‘Projektmanagements‘?

Kritiklosigkeit bzw. mangelndes "in Frage Stellen"?

Oder werden "Moden" auch in Ingenieur-Wissenschaften Einzug halten? ("Alle benutzen es, also benutzen wir es auch"-Denkmuster).

Scrum bietet definitiv einige Vorteile. Die Produktivität ist eine von ihnen: Die Entwickler verschwenden weniger Zeit, wenn sie täglich über den Status zu berichten haben. Ich versuche jetzt schwer, andere Vorteile zu finden... Sorry; Nichts [N.B.: Nicht immer geht es der Produktivität besser.].

[Anmerkung R.K.: Nicht immer geht es der Produktivität besser.]

Der Product Owner ist nur ein Listenschreiber: Informationen und Entscheidungen in Bezug auf die Stories muss er sich von anderen (insbesondere Manager) holen.

Und sie kaufen überteuerte "agilen" Systeme mit allen Möglichkeiten an Berichterstattung und schreiben lange Listen von schlecht definierten Aufgaben. Manche Teammitglieder nehmen diese Listen sogar ernst, besonders die Scrum Master tun das.

Scrum, die bekannteste Form von agilen Methoden ist ein regulierter Prozess, ebenso wie das traditionelle Wasserfall-Modell.

Für Scrum existiert kein Risiko und Issue-Management, kein Kostenmanagement, kein Vertragsmanagement, kein Konfigurationsmanagement, keinen Aufbau einer Projektorganisation...

[Anmerkung R.K.: Ist auch richtig so, denn Scrum ist nicht dazu da: Scrum ist nicht für Projektmanagement gemacht und somit dafür kaum geeignet.]

Fokussierung auf 100% Auslastung (zum Beispiel: "Lass uns eine Datenbank Story nehmen, so hat auch Bob etwas zu tun").

[Anmerkung R.K.: Das ist sowohl ineffektiv als auch ineffizient.]

Es bringt nichts wenn die Dinge "Scrums" genannt werden. Das letzte Mal das ich das Wort "Gedränge" hörte, war es im Zusammenhang mit Rugby in der Schule. Das bestand darin, ein paar meiner verschwitzten Mitschülern (und leider nur Jungen, nicht Mädchen) zu umarmen, um einen Ball kämpfend der irgendwo im Schlamm stecken geblieben war. Ein Ball der, muss ich ehrlich sagen, mich nicht wirklich interessierte.

In der Tat, Scrum "richtig" getan, gemäß Scrum, ist gar nicht so agil. Scrum ist nicht dazu entworfen, so wie es gelehrt wird, es zu ändern. Das ist ein massives Versagen. Es bricht das wichtigste agiles Prinzip.

[Anmerkung R.K.: In Wirklichkeit werden 3 Prinzipien verletzt: Flexibilität, Innovationsfähigkeit und Anpassungsfähigkeit.]

Die Idee, ein Scrum Master kann einen Projektmanager ersetzen, ist mangelhaft, aber sehr angenehm zu denken. Da einfacher.

[Anmerkung R.K.: Genau genommen, hat ein Scrum Master nur ca. 10% der Aufgaben des Projektmanagers. Folglich gehen etwa 90% der Aufgaben des Projektmanagers baden... oder sollen irgendwie im Verborgenen erledigt werden.]

- Sprints sind allzu starr
- Viel Planung inmitten der Arbeit
- Entwickler, die planen, (ist nicht deren Job!)

Die Dailys entarten zu Selbstdarstellungsshows, bzw. zu Rituale wo bestimmte Art der Kommunikation stattfindet, je nach Team, und bestimmte Themen werden (im Konsens) verschwiegen. Das bringt nicht viel.

Für den einen, dessen Tag nach einer Stunde langen Stand-Up starten wird, in dem er für vielleicht 5 Minuten gesprochen hat, ist der Stress vorprogrammiert. Es wird mindestens eine Stunde dauern, eine Tasse Kaffee benötigen und ein bisschen zufällig im Internet surfen, bevor man mental für die Codierung vorbereitet ist.

Ich habe den Eindruck, dass Manager sich auf Scrum freuen, weil Scrum "Last" (Arbeit) und nicht zuletzt Verantwortung vom Management wegnimmt.

Nach einigen Sprints, wenn wir uns ein Release des Produkts ansehen, fühlt sich die Architektur irgendwie wie zusammengeschustert, eher als mit gutem Design nach besten Wissen und Gewissen erstellt, welches uns ein flexibles, zuverlässiges Produkt ergeben soll. Sicherlich besagt uns die Theorie, dass wir mit "Refactoring" im Laufe jedes "Sprints" die zugrunde liegende Architektur anzupassen vermögen, aber hätten wir das gemacht, dann würden wir unsere gesamte Zeit damit verbringen die ganze Architektur zu überarbeiten anstatt die eigentlichen Funktionen zu implementieren.

[Anmerkung R.K.: Auch für das Refactoring der Architektur könnte ich eine "O-Notation" einführen.(Siehe **O-Notation**). Ganz neue Forschungsfelder der SW-Engineering tun sich den begabten Informatiker auf!]

Teams die glauben, dass es wichtiger ist, "mehr Arbeit" zu tun, als Stories zu Ende zu machen.

- Zu viel Brute-Force Story-Grooming, verursacht durch zu wenig funktionales Prototyping und zu wenig Business Systemanalyse und zu wenig Design-Techniken, die Modelle "cross-checken" könnten.
- Mangel an Modellen, dadurch zu wenig Story-Kontext und undefinierten Systemgrenzen.
- Übermäßiges Vertrauen auf Nutzer um zu wissen, was Wert liefern würde.
- Scrum verfällt zu Kanban... vielleicht keine schlechte Sache.

- Time-Boxing ist kontraproduktiv. Entweder ganz oder gar nicht.
- Wenn priorisiert, soll das im Vorfeld erfolgen.
- Der einzige Vorteil von Time-Boxing ist es, Leute, die stundenlang labern (sich selbst beweihräuchern) zu stoppen. Das funktioniert aber nur bei Meetings.

Scrum über vereinfacht mit Comicstrip-artigen Bilder (das könnte zu den vielen nicht Entwickler passen, die zwar entwickeln aber sonst kaum eine Ahnung haben).
Nicht nur dass diese beiden "agilen" Prinzipien vorhandene Beweise und einfache Logik trotzen, sondern dass sie auch eine Art Kindergarten-Mentalität fördern. Ich habe noch nie gesehen, dass man Scrum bei Design, Buchhaltung oder Ein- und Verkauf benutzt: oberflächliche, Bilderbuch-Ähnliche Vereinfachungen, wie die Sache mit den "Story Points" (Bewertung der 'Geschichten').

Schätzen (anstatt zu berechnen) fördert die Verantwortungslosigkeit - Besonders dann, wenn Teammitglieder schätzen, die erwartungsgemäß keine Ahnung haben (müssen keine haben, da in einem anderen Feld tätig).

Natürlich ist es auch besonders weltfremd wenn man von allen Teammitglieder erwartet, überall etwas zu wissen und einsetzbar zu sein.

Kein Software-Design ist vom Anfang an perfekt. Es gibt immer unvorhergesehene Veränderungen; manchmal durch das Unternehmen bedingt, manchmal durch technische Herausforderungen, manchmal durch nicht-funktionale Anforderungen. Ich glaube Scrum macht es schwieriger, Veränderungen durchzuführen. Es existiert stets diese Verpflichtung an Arbeit für alle, dazu jeden Sprint durch zu planen. Nicht immer ist das möglich. Manchmal ist eine Aufgabe schlichtweg zu groß um sie so aufzuteilen, dass sie in einem Sprint getan werden kann. Auch eine strikte Einhaltung der Sprint-Vorgaben zwingt manchmal etwas zu tun von dem man weißt, dass die Arbeit in einem späteren Sprint wegen einer Designänderung wieder verworfen wird. Dies kann sehr ärgerlich sein, und ich habe das mehrere Male erlebt.

Man behauptet immer wieder; Scrum in seiner "reinster" Form funktioniert. Doch, und auch wenn es stimmt und Scrum (theoretisch) funktionieren würde, die Realität sieht anders aus und Scrum ist Nonsens.

Es wird schlimmer. Wenn Sie Ihren Prozess anpassen um diesen Problemen zu entgehen, können Sie wegen "Scrumbut" angeklagt werden, das klingt irgendwie ungesund, und anscheinend ist eine fatale Schwäche. Sie können diese Probleme vermeiden, indem Sie einen "Certified Scrum Master" bezahlen. Es gibt eine ganze Industrie von Scrum-Berater, die versuchen, Ihre nicht-agile Organisation agiler zu machen, indem Sie die Gedränge-Zauberei anwenden. Doch seltsamerweise haben diese Master es verboten bekommen auf Code direkt zur Arbeit oder Verantwortung um irgendetwas zu liefern zu übernehmen.

Ich habe Scrum Trainings besucht, ich habe mit mehreren Scrum Masters an verschiedenen Projekten gearbeitet. Ob diese Projekte erfolgreich waren oder nicht, hing von der Mannschaft ab, nicht vom Scrum. Scrum-Techniken können sicherlich ein funktionierendes Team helfen, erfolgreich zu sein, aber sie können einem nicht funktionierendem Team nicht verhelfen nicht zu scheitern.

[Anmerkung R.K.: Wenn ich das richtig verstehe, werden erfolgreiche Teams mit Scrum auch Erfolg haben, während nicht erfolgreiche Teams auch nicht mit Scrum Erfolg haben werden. Klingt doch irgendwie einleuchtend...]

Vielleicht ist Scrum so gemocht weil viele IT-Fremder, insbesondere Absolventen der Sozial- und Kommunikationwissenschaften, sich hier "zu Hause" fühlen?

Scrum ist vielmehr als Framework zu verstehen, innerhalb dessen verschiedene Prozesse und Techniken zum Einsatz gebracht werden können. Das ist einer der Hintergründe auch für diese Aussage im Scrum Guide:
Scrum ist
1 Leichtgewichtig
2 Einfach zu verstehen
3 Extrem schwer zu meistern.

[Anmerkung R.K.: Zu:
1. Ja, ohne Zweifel
2. Ja, wie ein Spiel
3. Richtig, aber warum?

Hier möchte ich doch ein klein bisschen weiter untersuchen.
1. Leichtgewicht? Ja..., bis max. 10 Teammitglieder und ohne Interaktionen mit anderen Teams... (außer "Scrum von Scrums"...)
2. Einfach zu verstehen? Nein: Keiner versteht es. Es ist leicht **anzuwenden**, verstanden wird es nicht.
3. Warum ist es so extrem schwer zu meistern, wenn es so einfach anzuwenden ist? Hier liegt der Hund begraben. Wie kann man es "meistern" wenn man es nicht verstehen kann? Also, entweder meistern es nur diejenige, die es verstehen, oder keiner versteht es und ein "Meister" wird man zufällig, wenn Scrum "zufällig" funktioniert (und so lange wie es funktioniert).

Und weiterhin: Was braucht man, um es zu verstehen?
Vielleicht:
- Psychologie des Einzelnen und der Gruppe
- Kommunikationswissenschaften
- Gruppendynamik

- Welche Rolle spielt tatsächlich der Scrum-Master?
- Welche Interaktionen ergeben sich mit dem PO?
usw. usf.
Ich habe keine Ahnung. Ist mir ehrlich gesagt auch Schnuppe. Sollen sich die Sozialwissenschaftler damit plagen; mein Ziel ist ein Programm zu erstellen. Und dafür kann ich nichts gebrauchen, was dermaßen labil ist.]

Nun, der Job entartete zu Schätzungen und "Sprinten", es ging nicht um das Entscheiden, welche Lösung die beste wäre und sie in einer sinnbringenden Weise zu implementieren, vielmehr alles in Aufgaben Zersplittern, die weniger als 4 Stunden andauern sollen.
Ich war einfach verblüfft; diese starren geplanten Sprints, die Teams synchronisieren sollten, sind ein "One-Size-Fits-All" Fehler. Das Interessante aber war, dass das 'Management' darauf **bestanden** hat, dass es funktionierte.

Zu viel "Verwaltung" (*) - Zu wenig Programmierzeit (bzw. produktive Arbeit).
Probleme:
- Designen die SW immer aufs Neue mit jedem Sprint
- Daily: Das ganze Team steht ca. 15 Minuten (oder länger) um zu erzählen, was sie gemacht haben und was sie machen werden. Im besten Fall, wenn die Firmenkultur offen ist, erzählen sie auch welche Hindernisse sie haben
- Grooming (Das gesamte Team sitzt - oder noch schlimmer; steht - eine Stunde oder so um "Story Points" zu "schätzen")
- Sprint Planung: in zu kurzer Zeit entscheiden die Wenige des Teams, die Ahnung haben, was im Sprint geschehen soll. Der Rest steht und hört: unproduktive Zeit für die Meisten
- Sprint Retrospektive: Viel Palaver und oft irgendwelche "Aktivitäten" (m.a.W. Spielchen) die viel zu lange dauern und die Leute von der produktiven Arbeit abhalten.

[Anmerkung R.K.: (*) "Verwaltung" ist schlicht und einfach Verschwendung.]

Es wird behauptet, Scrum funktioniert aber keiner weiß genau warum. Es gibt Erklärungsansätze wie z.B. "gesteigerte Kommunikation". Sei es drum, eigentlich weiß keiner welche "Schrauben" was bewirken. Das ist Hokuspokus. Es ist in Wirklichkeit unwissenschaftlich und willkürlich.
Stellen Sie sich mal vor, Sie sind mit Ihrem Wagen in der Werkstatt und Ihnen wird beigebracht, bei Defekten mit einem Hammer an einer Stelle des Motors zu klopfen. Warum? Keiner weißt es, aber es soll funktionieren.

Scrum-Teams verbringen viel Zeit damit, herauszufinden wie man die Arbeitsbelastung neu ordnen kann, um die Burn-Down Grafik anstatt die Produktivität zu optimieren. Die schlechten Ergebnisse bezüglich der (echten)

Produktivität resultieren unter Anderen aus einer Verschiebung von Inhalt; Überprüfungen um unvollständige Arbeit doch als "fertig" (gemäß "Definition of Done") zu markieren, und Stories so unscharf zu beschreiben, dass man genügend Spielraum hat um sie doch zu schaffen (die Entwickler können die Story zu einem bestimmten Zeitpunkt als getan definieren, und nicht bei einer bestimmten Funktionalität).

[Anmerkung R.K.: Klingt echt wie Hokuspokus... Faktisch Verschwenderisch.]

Es ist kaum zu glauben: "agile" Methoden die "schreiben vor", dass man
- täglich Stand-Ups halten muss
- Groomings haben muss
- Sprintplanungen haben muss
- Sprints haben muss, mit immer wieder gleicher Dauer
- Retrospektiven haben muss
und vorschreiben dazu, dass Teams z.B.
- nur Generalisten haben müssen
- aus 4 (oder so) bis zu 9 (oder so) Personen bestehen müssen
usw. Und viele glauben das und machen mit.

Dann kamen die "Scrum Trainer" und mischten alles und alle auf. Die religiösen Fundamentalisten der Entwicklungswelt. ISIS mit einem Lächeln und einem MacBook.

Unser Arbeitgeber bezahlt diesen verkleideten Scharlatane viel Geld um diesem einfachen, flexiblen Prozess mit so viel zusätzliches Gepäck zu belasten, mit so viel zusätzlichen Jargon und mit so wenig spürbaren Verbesserungen der Produktivität, dass sie die ganze Sache ruiniert haben.

Und was ist mit all den Sitzungen? Was für eine Zeitverschwendung! Einige von uns fühlen sich nicht wohl, im Stehen in einem Raum voller Menschen zu reden. Das ist nun mal wie wir sind. Man soll mir einfach mal eine E-Mail oder eine Slack-Nachricht senden, wenn man etwas braucht, und ich werde gerne beantworten.

Unter Berücksichtigung all dieser zusätzlichen Verwaltung werden wir alle verlangsamt; zuvor produktive Entwickler wurden unproduktiv und das Prozess der Software-Entwicklung wurde zu einer Buzzwort-Reiterei, einem Mikro-management, einer gleichgeschalteten "Fabrikation". Kein Wunder, dass einige von uns Scrum hassen. Scrum-Trainer: Sie können Ihre Stories, Punkte, Burndowns und schwachsinnigen Planung Poker mitnehmen und sie sich sonst wo stecken. Arbeitgeber: vertrauen Sie das Wissen Ihrer Entwickler und was

sie tun; das ist doch der Grund, warum Sie uns angestellt haben.

Das Problem mit den meisten agilen Entwicklungsmethoden (und wenn ich das sage, meine ich Scrum, denn das ist, was die Menschen am Meisten verwenden) ist, dass es die Produktivität als vorhersehbar betrachtet wird. Es kann sich so anfühlen als würden Sie mehr tun und hätten Sie mehr Freiheit, doch die Wirklichkeit für viele ist, dass man Mikromanagement anwendet und weniger geschaffen wird - der Unterschied ist; es ist sichtbarer.

Die standardisierten Rituale erlauben den Einsatz von jungen, relativ unerfahrenen Beratern. Gerade zur Boomzeit von Scrum und den "agilen Methoden" standen wenig erfahrene "Experten" zur Verfügung. Junge "Berater" mussten rekrutiert werden und möglichst schnell Wert schöpfend eingesetzt werden. Dies war durch die Ritualisierte Form relativ einfach. Die Beratungsneulinge mussten lediglich die agile Ideologie und die standardisierten Ritualabläufe lernen, um dann beim Kunden für einen vierstelligen Tagessatz abgerechnet werden zu können. In einem Unternehmen ließen sich beispielsweise junge Hochschulabsolventen direkt als Scrum-Master einsetzen, weil der hohe Standardisierungsgrad der Rituale eine einfache Reproduktion (Wiederholung) der Kommunikationsinstrumente, Interventionsformen und Interaktionsmechanismen ermöglichte.

Aus Hans-Peter Korn; Die "agile" Organisation: Vom Hype zum Daily Business
Dominant im Jahr 2002 war XP. Heute, zehn Jahre später, dominiert Scrum (allerdings nicht in "Reinkultur" sondern primär in diversen Abwandlungen) die agile Landschaft. Das zeigen übereinstimmend drei unabhängige Studien aus dem Jahr 2012:
[SwissQ Agile Trends & Benchmarks 2012, online verfügbar in www.swissq.it/resources/detail/8]
[Swiss Agile Study 2012, online verfuügbar in www.swissagilestudy.ch/ und erste Ergebnisse in http://www.lean-agile-scrum.ch/wp-content/files/00_Swiss%20Agile%20Study%20Resultate.pdf]
[Status Quo Agile 2012, online verfügbar in http://www.hs-koblenz.de/Status-Quo-Agile.4782.0.html]

[Anmerkung R.K.: Was beweist, dass das reine Scrum keine Lösung ist, denn das meiste "Scrumbut" ist.]

Scrum meiner Erfahrung nach ist ein (Versprechungs-)Modell für das obere Projekt- und Produkt-Management im Allgemeinen, welcher angibt, ein Team effizienter und produktiver, auf das optimale Niveau, zu bringen. Die meisten 'Manager' der oberen Etage glauben dieser Werbung und starten durch.

Anmerkung: Das als "SAF" bekannte Framework (Scaled Agile Framework® SAFe®) ist nichts weiteres als eine "Scrum Sekte" (Scrum etwas anders angewendet) mit dem Zusatz von "Lean" als Buzzwort. Ziel ist es, Scrum auf größere Unternehmungen/Unternehmen anzuwenden (zu verkaufen).

Fazit aus meiner Sicht

Scrum kann anscheinend miserable Teams und unbedarfte "Entwickler" zu einer gewissen Produktion bewegen; wahrscheinlich aber bedingt dies entweder ein geringeres Arbeitsergebnis oder das Abdanken der besseren Entwickler, gute Entwickler leiden stillschweigend unter solchen Bedingungen und wursteln sich bestenfalls durch.

Die gröbsten "Scrum" Mängel

Nun gibt es einen Grund, warum Scrum so beliebt ist und warum es nie wirklich agil sein kann, zumindest nicht in der Weise, wie es so viele Teams verkauft wird.
Die meisten Menschen wollen in wirklich keinen agilen Prozess.
80% der Menschen, die Scrum oder andere Formen der agilen Methoden praktizieren, sind nicht wirklich agil und sie haben auch kein Interesse daran, agil zu sein. In Wirklichkeit sind sie in ihrem gebrochenen Prozess "unglücklich glücklich", weil Scrum (und jedes andere formalisierte "agiles" Prozess) ihnen etwas gibt, was sie tief wünschen.
Die Menschen wünschen sich eine Struktur, die die Verantwortung wegnimmt. Einen strukturierten agilen Ansatz gibt ihnen das. (1)

Denken Sie darüber nach; die Leute wollen ein Kochbuch, Rezepte, Tipps und Tricks, 5-Stufen-Prozeß, 12 "Hacks" und so weiter. Das "Kochbuch" wird schon sagen, was zu tun ist, so dass sie nicht selbst die Verantwortung für den Prozess nehmen müssen.

Wenn Sie Scrum verwenden und es geht schief, können Sie Scrum die Schuld geben. Das ist genau das, worum es in diesem Thread geht: "Es ist nicht unsere Schuld, der Prozess ist falsch! (2)"

[Anmerkung R.K.: (1) Sehe ich anders, die (in Deutschland zumindest) Mitarbeiter wollen Verantwortung übernehmen, doch Scrum nimmt de Verantwortung weg vom einzelnen und verlagert sie; zunächst zum Team, wo sie nichts zu suchen hat, im entwickelten Stadium wird sie unsichtbar.]

[Anmerkung R.K.: (2) Und nochmals denke ich anders; Der Rahmen (in diesem Fall Scrum) ist die Instanz, welche die (Rahmen-)Bedingungen stellt und somit die Struktur vorgibt, wenn Rahmen und Struktur nicht das halten, was sie versprechen, werden die Mitarbeiter es auch nicht können.]

[Anmerkung R.K.: "5-Stufen-Prozess" = Manifest? - "12 'Hacks'" = Agile Prinzipien? - "Kochbuch" = Scrum-Rituale?]

[Anmerkung R.K.: Und noch eine Sache fällt äußerst negativ auf: Die "Schuld" dafür, dass Scrum nicht funktioniert, liegt so gut wie immer bei den Mitarbeiter: entweder a) falsche Kultur und Denke, oder b) falsch angewendet (wobei es angeblich so leicht anzuwenden ist) oder c) weil die Mitarbeiter keine Verantwortung tragen wollen.]

Scrum wird eingesetzt weil es einfach ist und man damit schnell und relativ schmerzlos angeben kann: "Wir sind agil". Man kann Scrum anwenden ohne sich die geringsten Gedanken gemacht zu haben, was eigentlich "Agil" überhaupt ist, seine Vor- und Nachteile, und was man überhaupt damit erreichen will (außer anzugeben).
Die Rechnung wird, meiner Meinung nach, nachher kommen, wenn die Mitarbeiter nach ca. 60% Planungs- und Verwaltungsaufwand (in jedem Sprint) ein mickriges Ergebnis liefern.

Fazit aus meiner Sicht

Scrum scheint eine Lösung zu sein, die immer noch auf der Suche nach ihrem Problem ist. Traurigerweise entstehen dadurch neue und überflüssigen Probleme.

Über Kanban

Kanban anerkennt vorbehaltlos die aktuell praktizierten Vorgehensweisen, Rollen und weitere Organisationsaspekte, macht sie aber sicht- und messbar und schafft damit objektive Voraussetzungen für schrittweise Verbesserungen auf Basis von 3 Grundprinzipien und 5 Kerneigenschaften:

[Anmerkung R.K.: Mit "Kanban anerkennt vorbehaltlos die aktuell praktizierten Vorgehensweisen" scheint mir, ist den Agilen einen "agilen" Lapsus unterlaufen: "Kanban" kann nichts "anerkennen", denn es handelt sich um

einem Begriff („kan“ und „ban“; Signal und Karte). Was wahrscheinlich gemeint ist, ist folgendes: "Wir Agilen haben es geschafft, einen Rahmen zu gestalten, der ein bisschen wie Kanban funktioniert, oder zumindest wie wir uns vorstellen können, dass Kanban funktioniert."]

Die 3 Grundprinzipien:

- Beginne dort, wo du jetzt bist
- Respektiere bestehende Prozesse, Rollen, Verantwortlichkeiten
- Vereinbare mit den anderen ein fortlaufendes (nie abgeschlossenes), inkrementelles, evolutionäres Veränderungsvorgehen.

Die 5 Kerneigenschaften:

- Fluss der Arbeit ("Workflow") visualisieren
- Work In Progress (WiP) begrenzen
- Arbeitsfluss messen und kontrollieren
- Regeln des Prozesses explizit machen
- Modelle als Mittel der Prozessverbesserung nutzen.

Daraus resultierende Prozessverbesserungen sind z.B.

- Unterschiedliche und für jedes Projekt, Team oder Produkt optimalen spezifischen Prozesse
- An den Arbeitsschritten können ein Team als Ganzes oder einzelne Spezialisten oder auch verteilte -Teams kombiniert mit Einzelpersonen transparent arbeiten
- Sowohl ein kontinuierlicher Arbeitsfluss als auch ein natürlicher Arbeitsrhythmus ohne "künstlich" aufgeprägte Timeboxes, Sprints oder Iterationen sind möglich
- Die Reihenfolge der Arbeitspaket-Erledigung ist kurzfristig festlegbar
- Die ausnahmsweise Verletzung der (Work in Progress) WIP-Grenzen (z.B. bei Notsituationen) ist als regulärer Prozess möglich
- "Swimmlanes" für verschiedene Serviceklassen erlauben spezifische Kapazitätszuordnungen und Durchlaufgeschwindigkeiten oder auch termingebundene Arbeiten
- Methoden der quantitativen statistischen Prozesskontrolle (Durchlaufzeiten, Verweilzeiten etc.) zur Prozessoptimierung sind anwendbar und erlauben ein gezieltes Controlling (Steuerung)
- Experimente zur Prozessoptimierung sind erlaubt.

[Anmerkung R.K.: Was im Auge bei den Punkten zum Kanban sticht ist zunächst, dass die Punkte, nicht wie bei "Agile", sehr konkret sind, einfach und klar. Darüber hinaus fehlt die "pompöse" Sprache wie gesehen bei "Agile" und XP.]

Kanban bedeutet "Signal-Karte" und hat mit dem, was "Agile" daraus gemacht hat, kaum etwas zu tun.

"Kanban at Work": Vor und Nachteile

Immerhin sehen Sie hier bei den Kanban Prinzipien, wie Scrum in Frage gestellt werden, und das ist schon ein Anfang.

Nachteile

- Zu viele Meetings, die aus "Scrum" geerbt wurden; das Problem liegt hier bei der praktischen Anwendung durch unbedarfte Teams, nicht an der Beschreibung von "Kanban".
- "Zettelwirtschaft at its best" (muss aber nicht notwendigerweise etwas schlechtes sein). Diese Zettelwirtschaft ist eigentlich auch eine "schlechte" Sitte oder Einführungsmanko; gehört nicht zum Konzept des Kanbans („Signalkarten", nicht -Story- Beschreibungszettel).
- Kanban ist einfach und beinhaltet an sich keine dogmatischen Ritualen wie z.B. Scrum; Stand-Ups und andere Unsitten sind von außen (Scrum) injizierte Pesten. Siehe **Kanban**.
- Kanban ist flexibel und kann gut angepasst und angewendet werden.

Fazit aus meiner Sicht

Kanban ist ein Versuch von Scrum weg zu kommen. Hier hat man einen Begriff der Serienproduktion (stammt von der Firma Toyota, PKW Serienherstellung) teilweise auf Projekten angewandt und mit "agilen" Ideen vermischt. Das Beste von beiden Welten, könnte man glatt behaupten. Leider stimmt das nicht wirklich: Man hat unbedachter Weise bei Kanban noch viel Gift von XP/Scrum bei der Umsetzung mit übernommen.

Anmerkung: Wenn Sie achtsam die weiter oben aufgelisteten Grundprinzipien, die Kerneigenschaften und die daraus resultierenden Prozessverbesserungen lesen, dann wird Ihnen deutlich wie eng sich (dieses) Kanban eigentlich von dem "Toyota Way" für die Autoserienproduktion ableitet.

Kanban ("Signal-Karte") "schmiert" das Prozess, indem die Kommunikation standardisiert und vereinfacht wird. Doch Kanban "ist" kein Prozess, wie die "Agilisten" glauben machen wollen, und schon gar nicht eine Entwicklungsmethode.

"Lean" Prinzipien

Die sieben Prinzipien der Lean-Entwicklung:

1. Verschwendung vermeiden (siehe weiter unten: **Lean**)
2. Feedback einfordern (alle Teilnehmer einbeziehen - Kommunikation intensivieren)
3. Spät entscheiden (sich so viel Zeit wie möglich nehmen: zum Denken und zum Planen)
4. Schnell ausliefern (so schnell wie möglich die Entwicklung durchziehen, nachdem "entschieden" wurde)
5. Team trägt Verantwortung (impliziert Selbst-Organisation)
6. "Integrität einbauen" ("stets" standardisieren, so wie ich es verstanden habe)
7. Die Gesamtheit optimieren (den gesamten Prozess verbessern)

Andere "Übersetzungen" postulieren sinngemäß:

- Die Arbeiten gleich beim 1. Mal richtig machen (inhaltlich zum 1. Punkt: keine Verschwendung)
- Den Arbeitern erlauben, Entscheidungen zu treffen (nicht nur "ganze Teams" sondern auch auf persönlicher Ebene)
- Stets verbessern (inhaltlich wie beim 7. Punkt)
- Das Lernen unterstützen (entfernt verwand vielleicht mit dem 2. Punkt).

Ohno (1988) verwendet das japanische Wort "Muda" um Verschwendung (**Lean**) zu identifizieren und definiert sie als:

- Defekte (Einheiten oder Stücke),
- Überproduktion von nicht benötigten Waren,
- Vorrat von Ware die auf die nächste Verarbeitung oder zum Gebrauch warten
- Unnötige Verarbeitung,
- Unnötiges Transport (von Menschen),
- Unnötiges Transport (von Ware) und
- Wartezeiten (Mitarbeitern, die auf einer Umrüstung oder auf einer Aktivität eines anderen Teams warten).

Womack und Jones (1996) stellen folgende Beispiele von Muda:

- Fehler, die verbessert werden müssen
- Herstellung von Gegenastanden, die niemand will
- Verarbeitungsschritte, die nicht benötigt werden
- Bewegung von Ressourcen ohne Zweck
- Mitarbeiter warten auf Lieferungen

- Defekte Ergebnisse
- Waren und Dienstleistungen, welche die Bedürfnisse des Kunden nicht erfüllen.

Und nun kann man versuchen, diese Verschwendungen auf die SW-Entwicklung zu übertragen. So kommen wir zum nächsten Punkt:

LSD: Lean Software Development (Schlanke SW-Entwicklung)

Die Lean Software-Entwicklung ist auf fünf Grundprinzipien aufgebaut, diese gestalten die Software-Entwicklung irgendwie besonders. Hier die Prinzipien:

- Wert aus der Sicht des Endkunden spezifizieren
- Identifizieren Sie alle Schritte in der Wertschöpfungskette die Wert schaffen, beseitigen Sie alle Schritte, alle Aktivitäten und alle Gewohnheiten, die keinen Wert schaffen
- Lassen Sie die verbleibenden Schritte in einer engen und zusammenhängenden Weise ablaufen; damit das Produkt (zum Kunden) reibungslos fließen kann
- Sobald das Prozess fließt, lassen Sie (den Kunden) Wert von der nächsten Upstream-Aktivitäten entnehmen
- Da diese Schritte zu mehr Transparenz führen, können Manager und Teams weitere Verschwendung identifizieren, somit kann ein kontinuierliches Verbesserungsprozess eingeführt werden.

Und hier die 7 Verschwendungen der SW-Entwicklung:
1. Überproduktion (Zusätzliche Funktionalität, "Vergoldung")
2. Inventur (Anforderungen, Verwaltung)
3. Zusätzliche Produktionsschritte (mehr Komplexität als notwendig, überflüssige Funktionalität)
4. Bewegungen (Suche von Informationen)
5. Fehler (Bugs, Fehler in der SW, Tools mit Fehler)
6. Wartezeiten (Warten auf Entscheidungen, Warten auf Kunden-Feedback)
7. Transport (Schnittstellen, Workflow der Arbeit).

Der Schwerpunkt der schlanken Entwicklung ist es also, die Verschwendung wahrend der Software-Entwicklung zu reduzieren

Das Gute an dem Lean Ansatz ist, aus meiner Sicht, dass das Ziel "Wert schaffen" ist. Und darauf kommt es an.

Fazit aus meiner Sicht

Wie bei Kanban sticht auch hier der Mangel an pompösen Wörter und die Verwendung von Begriffen, die einfach und klar sind.

Anmerkung: Wie bei Kanban, wenn Sie achtsam über die "Lean"-Konzepte nachdenken, so wird Ihnen deutlich wie sich auch "Lean" eigentlich aus der Autoserienproduktion vom "Toyota Way" ableitet.

An diesem Punkt der Untersuchung angelangt, möchte ich darauf hinweisen, dass "Agile" und "Lean", wenn Sie die Definitionen durchlesen, sehr wohl einige Unterschiede aufweisen: **Agile postuliert Ansätze, die mit Lean nicht kompatibel sind.** Trotzdem wird oft "Lean" mit "Agile" vermischt. Mir persönlich ist es aber wichtig beide sauber getrennt zu halten, ganz einfach aus dem Grund, dass sie eindeutig nicht zusammenpassen.

Und noch auf einem Punkt möchte ich hier aufmerksam machen: Nicht nur von der Definition unterscheiden sich beide (Agile und Lean) nicht unwesentlich, sondern auch bei der praktischen Anwendung existiert eine Riesendifferenz:

Agile: ist eine Wohlfühlmethode wo das 'Management' IT-Berater bestellt und viele hübschen Begriffe (Buzzwörter) benutzen darf

Lean: bedeutet harte Arbeit: Das Management ist angehalten, zusammen mit den Arbeitern Mehrwert zu schaffen und an das echte "Ding" wird operiert (z.B. Prozesse ändern, Verbesserungen einführen, Verschwendung ausmerzen, Wert des Produzierten definieren und messen u.a.).

Über Projektarbeit

Die "agile" Entwicklung, so wie sie definiert ist, ist keine Projektarbeit, denn da finden Sie keine Berücksichtigung der Zielorientierung nach:

- Zeitgrenzen
- Umfang
- Kosteneinhaltung

die ein Projekt ausmachen.

Die wichtigste Funktion des IT-Projektmanager ist es, zu verschiedenen Aktivitäten Ressourcen zuzuordnen. Darüber hinaus trägt der Projektmanager Verantwortung für die Projektplanung und -schätzung, Steuerung, Organisation, Vertragsmanagement, Qualitätsmanagement, Risikomanagement, Kommunikation und Personalmanagement (HR).

[Anmerkung R.K.: Das ist zu viel (des Guten). Kein Wunder, dass ein Projekt-manager ein bisschen mehr von den "agilen" Methoden haben möchte.]

Warum Projekte scheitern

Hier einige Punkte:

- Fehlendes/falsches Systemdesign
- Mangelhafte Projektplanung
- Fehlende Managementunterstützung.

Zu den häufigsten Faktoren:

- Unrealistische oder unbeschriebene Projektziele
- Ungenaue Schätzungen der benötigten Ressourcen
- Schlecht beschriebene Systemanforderungen
- Mangelhafte Berichterstattung über den aktuellen Status des Projektes
- Ignorierte Risiken
- Schlechte Kommunikation zwischen Kunden, Entwickler und Anwender
- Die Verwendung von unreifen Technologien
- Unfähigkeit die Komplexität des Projektes zu handhaben
- Mangelhafte Entwicklungspraktiken
- Schlechtes Projektmanagement
- Nicht mitziehende Stakeholder
- Kommerzieller Druck (Budgetierung).

Vor einigen Jahren, schätzten wir (grob), dass ein Projekt (neues Geschäftsfeld) 2 Mann-Jahren brauchen würde. Das Management zählte dann **jeden** im Raum (6 Personen) als Vollzeitentwickler und stellte fest, dass die Arbeit also in 4 Monaten fertig werden sollte (zusätzlich also zu unserer täglichen Arbeit, natürlich).

Ich schlug vor, wir könnten alle im Büro zum Programmierer umtitulieren, dann würden wir in der Nächsten Woche fertig sein können. Irgendwie war ich aber der Streitsuchender!

Man kann einen Schuldigen fast überall finden, doch für mich ist es in erster

Linie das Management, das die Prozesse nicht versteht.

Ja, einen detaillierten Arbeitsplan und eine gründliche Analyse sind erforderlich. Jedoch sollte das geschehen, bevor das Projekt tatsächlich beginnt!
Ich fand zu viele Unternehmen, welche dies alles als Teil des Projektes betrachteten und sie wollten alle als Erstes eine Abschätzung (wobei das Ziel noch gar nicht präzisiert und über Änderungen erst gar nicht gesprochen wurde).

Fazit aus meiner Sicht

Kaum einem der Faktoren, die ein Projekt scheitern lassen, hat mit einer Entwicklungsmethode zu tun. "Mangelhafte Entwicklungspraktiken" könnte ich nehmen, hat direkt mit einer "Methode" jedoch wenig zu tun, sondern damit, "wie" man entwickelt (so wie ich das sehe).

Das heißt, wenn ich mich auf der Suche mache nach einer Methode oder Modell, dass das Scheitern von Projekten adressiert (minimiert), so muss ich die Liste hier (Faktoren) berücksichtigen.

Die Risiken bei Projekten

Aus den "Top Ten" Listen von:
- Addison
- Addison and Vallabh
- Böhm
- Han and Huang
- Schmidt et al. (Hong-Kong)
- Schmidt et al. (Finnland)
- Schmidt et al. (USA)

kann ich eine Liste der "gefährlichsten" Punkte (Risiken) erstellen. Die Reihenfolge ist stimmig:

1. Anforderungsmanagement:
- Unklare Anforderungen
- Falsch verstandene Anforderungen
- Systemanforderungen nicht ausreichend identifiziert.

[Anmerkung R.K.: Ich denke alle drei Punkte sind gleich wichtig.]

2. Ständige Änderungen (Siehe **Änderungen**.)
- Sich ständig ändernde Anforderungen; Umfang und Ziele ändern sich zu schnell

[Anmerkung R.K.: Problem auch als "moving targets" bekannt.]

3. Benutzer Engagement bzw. Einbeziehung des Benutzers
- Mangel an angemessener Beteiligung der Nutzer
- Mangelnde Zusammenarbeit von Benutzern.

[Anmerkung R.K.: Es ist nicht immer leicht, den Benutzern zur Zusammenarbeit zu bewegen; oft haben sie weder Lust noch Zeit dafür.]

4. Mangelhaftes Projektmanagement
- Unrealistische Zeit- und Kostenschätzungen
- Unzureichende Schätzung der erforderlichen Ressourcen
- Keine Planung oder unzureichende Planung (Schlechte Projektplanung)
- Projektfortschritt nicht genau genug überwacht
- Fehlen einer effizienten Projektmanagementmethode.

[Anmerkung R.K.: Hier sind mehrere Punkte zu finden, also sind hier leider auch mehr Risiken zu erwarten...]

[Anmerkung R.K.: Dieser Punkt:
- Entwicklung der falschen Benutzeroberfläche
kann sowohl an den Anforderungen als auch beim Projektmanagement liegen.]

5. Mangel an Skills
- Fehlendes Personal (Krankheit, gekündigt etc.)
- Mangel an erforderlichen Kenntnisse oder an Fähigkeiten der Projektmitarbeiter.

[Anmerkung R.K.: Bei den folgenden 2 Punkten
- Mangel an E-Commerce-Projekterfahrung, und
- Im Projekt ging es um die Nutzung neuer Technologien
könnte ich das Problem auch beim Projektmanagement suchen, also Punkt 4. Mangelhaftes Projektmanagement.]

6. Fehlende Unterstützung durch das Management/Unternehmen

- Mangel an Engagement und Unterstützung durch das Top-Management für das Projekt
- Abwesenheit der angegebenen Geschäftsvorteile
- Unternehmenspolitik mit negativen Auswirkungen auf das Projekt.

[Anmerkung R.K.: Nun, das sind leider Risiken, welche außerhalb des Projektes liegen. Man kann nur versuchen, früh genug das Problem zu identifizieren und eine Lösung anstreben, und wenn diese Lösung ein Ende mit Schreck sein sollte.]

7. Falsche Funktionalität und "Vergoldung" (Zu viel des Guten: Software aus "Gold" erstellen)

- Entwicklung der falschen Software-Funktionen
- Unzureichende Sicherheitsmerkmalen des Systems (unzureichende Sicherheit)
- Echtzeit-Performance mangelhaft
- Integrität und Verfügbarkeit der Datenbank.

[Anmerkung R.K.: Diese Punkte fallen aber auch indirekt im Bereich des Anforderungsmanagements und des Projektmanagements.]

8. Outsourcing

- von außen gelieferten Komponenten fehlerhaft
- von extern durchgeführten Arbeiten fehlerhaft.

[Anmerkung R.K.: Diese Punkte fallen aber auch indirekt im Bereich des Anforderungsmanagements und des Projektmanagements, speziell Schnittstellenbeschreibung.]

Fazit aus meiner Sicht

Es sieht sehr danach aus, als würden (mindestens) 99% der Projekten aufgrund Anforderungsmanagement oder Projektmanagement scheitern.

Weiterhin, da ich immer noch auf der Suche nach einer Methode (oder Modell) bin, welche das Scheitern von Projekten mit adressiert, so muss ich auch diese Liste hier berücksichtigen und nicht dem “Glauben“ unterliegen, eine Entwicklungsmethode das leisten kann, was die Agilisten so versprechen.

Über Mythen

Es gibt einige Konzepte die weit mehr Verwirrung stiften als die schlichten "Schlagwörter". Das sind die Mythen: diese Mythen, unermüdlich wiederholt, klingen vernünftig und das 'Management' will gerne daran glauben, weil sie (die Mythen) einerseits so "modern" sind, andererseits tun sie so wunderbare Sachen versprechen, das man sich ihrer magischen Anziehungskraft nur mit echter geistigen Anstrengung entziehen kann. (**Mythen**).

"Mitarbeiter können und sollen motiviert werden"
(**Arbeitermotivation**.)
Mitarbeiter-Motivation gibt es nicht. Wir haben es hier mit einem Wunschdenken zu tun: Mitarbeiter kann man nicht motivieren; die Entwicklung an sich ist das Ziel.

Dazu ein Text: "Später in den 60er Jahren begannen Pioniere wie Frederick Herzberg zu untersuchen, wie man die Produktivität verbessern kann; diese Pioniere versuchten die Motivation der Arbeitnehmer zu erhöhen indem man sie mehr im Entscheidungsprozesse teil nehmen ließ. Dies wurde 'Arbeitsbereicherung' genannt."

[Anmerkung R.K.: Und man hat gehofft, die Mitarbeiter wären einfältig genug, das mit der "Bereicherung" zu schlucken. Eine Weile vielleicht, aber nicht für immer.]

[Anmerkung R.K.: "Motivation" ist eine Fata Morgana: kann nicht erreicht werden. Grund: Symptombekämpfung. Siehe folgendes:]

Das man Mitarbeiter motivieren kann ist auch eine Art Wunschdenken, wobei "Pioniere" den Selbstbetrug unterlagen, dass die (Arbeits-)Welt ihre persönlichen Vorstellungen zu folgen hätte.

Sollte die Unternehmensleitung seine Arbeiter motivieren "müssen", so muss sich die Unternehmensleitung selbst an die Nase fassen: Sind die Arbeitsbedingungen so mies, die Arbeit so chaotisch, dass man die Mitarbeiter zum Arbeiten "motivieren" muss? Hier wird am falschen Ort angesetzt: Die zu verrichtende Arbeit muss verbessert werden, so dass sie leicht von der Hand geht. Das erfordert aber natürlich mehr Denkarbeit; klar es ist einfacher, externe "Coaches" und Unterhalter zu bezahlen (können Sie sogar von der Steuer absetzen). Diese sind aber nicht die Lösung.

[Anmerkung R.K.: Auf gleicher Weise werden lustige "Aktivitäten" (Spielchens) an (verspielten?) Mitarbeiter angeboten in der Hoffnung, diese würden danach "motivierter" mehr produzieren. Ein Zusammenhang ist nicht erkennbar, vielmehr werden Gelder verpulvert und Zeit vergeudet.]

"Community of Practice" oder "Wir bilden einen Komitee, um das Problem zu lösen" oder "Kreise (oder Boards etc.) finden Lösungen"
Nein! Sie liefern geschriebenes Papier am Besten mit vielen bunten Diagramme (in der Hoffnung, diese werden zumindest angeschaut) die danach keiner achtet. Es ist nur ein Zeitverlust, eine Arbeitsbeschaffungsmaßnahme, vielleicht kommen sich die Teilnehmer (sehr) wichtig vor. Aber die meiste Zeit wird nur gedrucktes Papier produziert, das anschließend in die Bedeutungslosigkeit wandert.

"Bottom-Up Intelligenz"
Das Herausfinden wie man die Arbeit optimal durchführt ist eine Aufgabe, die sowohl dem Management als auch den Mitarbeitern auferlegt. Zu glauben, dass die Menschen, die die fragliche Arbeit verrichten, sie am besten gestalten können, ist kurzsichtig (nicht zu Ende gedacht). Irgendwie will man diese Arbeit als dermaßen komplex betrachten, dass das 'Management' nicht in der Lage ist, sich dabei etwas zu denken oder gar etwas auszudenken.

[Anmerkung R.K.: Es ist schlichtweg viel zu optimistisch zu erwarten, dass ein Entwickler nicht nur sein Programmierjob machen kann, sondern auch Risikoanalyse, Qualitätssicherung, Anforderungsanalyse, usw. (als eine Art Eier legende Wollmilchsau) mit erledigt.]

"Pair Programming verbessert die Programmierung"
Ich glaube, hier ist nicht mal Wunschdenken im Spiel; vielmehr die dumpfe Wiederholung von nicht nachdenkenden Vielredner. Oft wiederholt aber ohne Beleg der Wahrheitsgehalt (durch Studien?) Kann funktionieren einer Weile, wenn ein Junior über die Schulter eines Seniors glotzt, doch damit hat sich das. (Siehe **pair**.)

"Selbstorganisierte Teams schaffen mehr"
Das mit den "selbstorganisierte Teams" ist auch wieder ein Schlagwort ohne Substanz. (**selbstorganisierende Teams**.)
Manchmal erzielen Teams eine gewisse "Selbstorganisation", manchmal nicht. Es kommt darauf an, so wie ich es sehe, auf die "soziale Organisation", die das Team darstellt. Eine Selbstorganisation kann besser sein, muss aber nicht. Und eine Studie die belegen könnte ob tatsächlich sich selbst organisierenden Teams besser sind oder nicht wird sehr lange auf sich warten lassen, denn eine Vergleichsmöglichkeit so gut wie ausgeschlossen ist (schon die Definitionen sind überaus unscharf, z.B.: Wann genau organisiert sich ein Team selbst?).

"Transparenz. Wir müssen regelmäßig wissen, was tatsächlich geschieht um wirksame Entscheidungen zu treffen"

Leute mit dem Konzept der 'Transparenz' scheinen zu glauben, dass jeder Mitarbeiter ohne Rücksicht auf Ausbildung, Erfahrung, Interessen, Nachdenkzeit, IQ oder emotionale Intelligenz in der Lage ist, 'wirksame' (hoffentlich gute, das ist es doch, worauf es ankommt) Entscheidungen zu treffen. Langsam frage ich mich: Ist das 'Management' dermaßen unfähig, dass jeder Hergelaufene seine Arbeit übernehmen kann?

[Anmerkung R.K.: Wir arbeiten in Projekten wohl oder übel in Umgebungen voller Unbekannten. Von Transparenz zu reden ist milde ausgedrückt wirklichkeitsfern. Entscheidungen zu treffen ist nicht leicht und man läuft bei diesen Ansätzen die Gefahr schnelle (und falsche) Entscheidungen zu treffen aus dem einfachen Grund, dass die Entwickler einfach entwickeln wollen und so wenig Zeit wie möglich mit Planen und Designen verbringen wollen, geschweige denn, Entscheidungen in unscharfen Umgebungen treffen zu müssen.]

"Scrum funktioniert als Methode für Teams in ganz bestimmten Grenzen"
Ein Scrum-Team hat 5 bis 10 Personen. Weniger, und Sie brauchen diese Struktur gar nicht. Mehr, und Sie können das Team nicht als "selbstverwaltetes Team" laufen lassen. Leider erfordern Software-Projekte oft einen größeren Aufwand. Die Antwort von Scrum auf diese Frage ist, Teams in die "richtigen" Größe zu teilen, und diese wiederum dann hierarchisch in einem "Scrum von Scrums" zu organisieren. Das ist so wie die alte Idee der kommunistischen Zellen (russischen Bolschewiki selbst organisiert in "Zellen" von etwa 10 Teilnehmer), die sich irgendwie "spontan" selbst organisieren würden (eine Art von sich selbst verwaltendes Arbeiterparadies). Das kann nicht funktionieren, vor allem, weil eine auf Kommando- und Kontrollorganisation (wie unsere Unternehmen heutzutage) von Natur aus hierarchische Organisationen sind.

[Anmerkung R.K.: Scrum funktioniert nicht immer und auch nicht immer öfters. Siehe weiter oben: „Über Scrum“.]

"Shipable Product Increment" (Lieferbares Produktinkrement)
Die Idee, dass man nutzbare SW liefert nach jedem Sprint (Iteration, Timebox) ist realitätsfremd, eine Art "Buzzwort" auch, und alle, die in Projekten arbeiten, wissen das: Eine SW die unfertig ist und womöglich wesentliche Bestandteile und Funktionalitäten fehlt, ist für den Kunde schlichtweg wertlos. Geschweige denn, sie beinhaltet Fehler, welche Menschen gefährden, Geldbeträge verschwinden lassen oder Ähnlichem. Das man solche "Produkte" liefern kann ist Nichts weiter als Wunschdenken (außer man liefert ein bisschen Text auf einer Webseite).

"Test Driven Development (TDD) lässt weniger Fehler entstehen"

TDD ist in Allgemeinen eine grobe Fahrlässigkeit durch Vereinfachung, kann aber natürlich funktionieren wenn die Aufgabe des Programms einfach genug ist. Die Tests des TDDs checken eine Anwendung gegenüber die (sich hoffentlich nicht ständig ändernden) Anforderungen. Das ist für den Entwickler gut, denn er kriegt eine unmittelbare Rückmeldung, doch mit QS (Qualitätssicherung) und Risikobewertung hat das leider sehr, sehr wenig zu tun. Es können sogar gravierende Fehler entstehen, wenn der Entwickler den Blick für das Ganze verliert weil er auf die vereinfachten "Tests" fokussiert, die gerade abgelaufen sind.

Über Regeln innerhalb einer Organisation

Regeln schaffen Ordnung. Darüber hinaus sollen die Regeln Sinn machen und nachvollziehbar sein, und zwar für alle Mitarbeiter der Organisation. Willkürliche Regel sind zu vermeiden. Zu viele Regeln sind ebenfalls zu vermeiden.

Klare Regeln geben den Rahmen für die Arbeit und somit Sicherheit. Das ist besonders bei Projektarbeit sehr wichtig.

Siehe **Regelwerk**.

Über Qualität

Qualität ist ein merkwürdiger Begriff: Alle wissen was es bedeutet, doch keiner kann es wirklich definieren. Sehen wir uns mal die beide nächsten Definitionen (Aus "ISTQB/GTB Standardglossar der Testbegriffe"):

(1) Der Grad, in dem ein System, eine Komponente oder ein Prozess die Kundenerwartungen und -bedürfnisse erfüllt [nach IEEE 610].

(2) Der Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt [ISO 9000:2000].

In beiden fällen geht es um "den Grad". Was auch immer das bedeutet. Ein "Erfüllungsgrad" von inhärenten Merkmalen ist auch nicht viel leichter zu messen.

Kann man Qualität messen? Eher weniger, man kann aber Risiken bewerten; auch über diese Bewertung kann man Qualität definieren: "Je mehr Qualität, weniger Risiken". Das ist auch nicht wirklich zielführend, denn eine SW die nichts tut bergt auch keine Risiken.

Kundenerwartungen sind schwer zu messen, ebenfalls Kundenbedürfnisse: Wie wir gelesen haben, wissen die Kunden oft gar nicht, was sie wollen.

Doch Qualität ist wichtig: ganze Unternehmen sind auf Qualität aufgebaut (Rolls Royce). Qualitätsprodukte haben die japanische Wirtschaft zur Weltspitze verholfen.

Zu einfach macht man sich es, wenn man die Qualität aufs "Testen" reduziert: So als würde man Autofahren aufs "Gas geben" und "bremsen" reduzieren und man sonst alles Andere außer Acht lassen würde (Verkehrszeichen, andere Teilnehmer, Erfahrung, Gepflogenheiten etc.).

Anscheinend mit dem Einzug der "agilen" Methoden wurde der Auszug der Qualität aus dem (sogenannten agilen) Projektgeschäft eingeläutet: Nicht nur, dass plötzlich Entwickler testen müssen, sondern es wird auch nahegelegt, dass Tester entwickeln. Für "alte" Programmierer ist das als "Mischmasch" bekannt; das geschah, wenn man z.B. versucht hatte einer "Integer" Variable einen reellen Wert zuzuweisen.

Doch das Testen bedeutet bei Weitem nicht Qualität sichern, sondern nur testen!
Beide Tätigkeiten werden immer noch geflissentlich verwechselt.

Testen ist dazu da die SW (Programme) zu checken, ob sie das machen, was der Kunde erwartet und hoffentlich spezifiziert hat.
Die Qualität zu „sichern“ aber zielt darauf die Maßnahmen zu treffen, (und diese am Besten schon beim Start des Projektes, Maßnahmen welche außerdem auch während des Laufes des gesamten Projektes durchgeführt werden sollen), die angebracht zu sein scheinen um zu gewährleisten, dass die Güte/Qualität der SW "sitzt".

Es ist demnach nicht das Wichtigste viel zu testen, sondern Qualität im Produkt "einzubauen". Darüber hinaus sind Maßnahmen, welche zu einer Verbesserung z.B. der Arbeitsbedingungen führen, auch ein Teil des Qualitätsmanagements.

[Anmerkung R.K.: Qualität müssen Sie so gut und so früh wie möglich planen.]

Über Unit- und automatisierte Tests

Es ist zu beachten das Systeme, unabhängig von ihrer Fähigkeit, Unit-Tests zu bestehen, doch fehlerhaft sein werden.

[Anmerkung R.K.: Unit Tests sind nicht verlässlich und insbesondere kein Qualitätsmaßstab. Das Gleiche gilt für automatische Tests: Diese lassen erkennen, das die SW es "noch tut"; das bedeutet streng genommen nicht, dass sie nicht schlimmer geworden ist.
Automatische Tests sind nur ein Tool um annehmen zu dürfen, dass die SW "noch" funktioniert.
Sehr oft aber handelt es sich bei den automatischen Tests um Verschwendung: erstmals müssen sie entwickelt (und noch schlimmer; gewartet) werden und zweitens zwingen sie den Entwickler, sich um (unwichtige) Issues zu kümmern die vielleicht sogar (oft) keine sind.
Und noch ein Punkt: Die Tester müssen sich immer wieder überzeugen, dass die Tests auch "testen", d.h., dass sie fehlschlagen wenn sie fehlschlagen sollen (das wird leider oft vernachlässigt).
Letztendlich sollen Sie die Tatsache nicht unberücksichtigt lassen, dass eine SW-Entwicklung bzw. eine Projektarbeit (der normale Zustand ist sehr "buggy") nicht wirklich mit einer Serienproduktion (der normale Zustand soll fehlerfrei sein) zu vergleichen ist, wo automatische Tests doch von unschätzbarem Wert sind. Der Grund dafür ist doch klar, oder?]

Weitere Probleme aus Sicht der SW-Entwicklung

Projekte dauern zu lange => Planen (richtig planen)
und
Projekte kosten zu viel => Planen (richtig planen)

[Anmerkung R.K.: Wobei noch ein nicht zu unterschätzendem Problem vorkommt; um das Projekt doch zu starten, müssen viel zu optimistischen Annahmen gemacht werden (sonst würde der Auftraggeber gleich das Ganze abblasen). Daher sind Anfangsschätzungen und -Planungen eher aus politischen Erwägungen so gefallen, wie sie sind, und die Stakeholder müssen damit wohl oder übel leben.]

Änderungen der Requirements (Anforderungen) => Änderungen sind zu minimieren

Probleme zu lösen:

1. Kommunikation/Informationsfluss: Ich bin der Meinung, dass es 2 unterschiedliche Sachen sind, werden aber in einem Topf geworfen.

2. Ein anderes Problem ist es, dass die Kunden ständig die Anforderungen ändern, bis in die letzte Minute.

3. Die Erstellung von Software, insbesondere von Unternehmenssoftware, ist ein komplexes Thema. Die Komplexität nimmt mit der Anzahl der involvierten Mitarbeiter und der Auswahl an Produkten bzw. unterschiedlichen Kundensegmenten überproportional zu.

[Anmerkung R.K.: Ein wichtiger Punkt ist es, die Komplexität (Verschwendung) zu reduzieren (nicht nur Herr werden, sondern regelrecht minimieren).]

Etwas Wissen zur Lösung der Probleme

Das Thema "Beyond Budgeting" hat sehr gute und interessante Ansätze. Meines Erachtens nach kann ich einiges zum Projektgeschäft / zur SW-Entwicklung hinüberretten.

Beyond Budgeting

(Aus Wikipedia)

Führungsprinzipien

- Werte: mit einigen wenigen, klaren Werten, Zielen und Grenzen lenken, nicht mit detaillierten Regelwerken
- Verantwortung: allen Mitarbeitern ermöglichen, selbstverantwortlich zu denken und unternehmerisch zu handeln; nicht die Befolgung von Plänen anregen
- Selbständigkeit: Teams die Freiheit und den Raum zum Handeln geben, ohne Mikro-Management von oben
- Organisation: ein schlankes Netzwerk aus ergebnisverantwortlichen Teams schaffen, keine zentralistische, funktional geteilte Pyramide
- Kunden: alle Mitarbeiter auf ihre Kunden ausrichten; nicht auf Hierarchie und Machtbeziehungen
- Transparenz: Information zum Zweck der Selbststeuerung offen

zugänglich bereitstellen, nicht den Zugang hierarchisch begrenzen oder Informationsmacht zulassen.

Performance-Management-Prinzipien

- Ziele: relative Ziele für kontinuierliche Verbesserung setzen, keine fixierten Leistungsverträge verhandeln
- Belohnung: gemeinsamen Erfolg basierend auf erbrachter Teamleistung belohnen; nicht einzelne Mitarbeiter durch Zielerreichung motivieren oder anreizen
- Planung: als kontinuierlichen und integrierten Prozess praktizieren, nicht als jährliches Top-down-Event
- Ressourcen: dann bereitstellen, wenn sie benötigt werden, nicht durch jährliche Zuteilung und Allokation
- Koordination: Zusammenarbeit marktüblich-dynamisch koordinieren, nicht über Planungszyklen
- Kontrolle: basierend auf relativen Indikatoren, Trends und Soll-Ist-Vergleichen, nicht mittels Planabweichung.

Ich denke, der Ansatz des "Budgets" nicht grundsätzlich falsch ist; dieser Ansatz kann durchaus helfen, die richtige... (es gibt keine richtige!) ...also doch schlicht eine optimale Lösung zu finden.
Einige Punkte sind wichtig:

- Mit klaren Werten und Zielen lenken
- Mitarbeiter tragen Verantwortung
- Netzwerk aus schlanken Teams organisieren.

Andere Punkte sind eher kontrovers, speziell:
Team-Belohnung: Man läuft die Gefahr, dass Gefühle der Ungerechtigkeit zwischen Teams entstehen.
Man kann nicht gerecht belohnen, schon gar nicht innerhalb eines Teams, denn dann werden zwischen den Teammitgliedern Ungerechtigkeiten (nicht alle leisten das Gleiche, sie müssen es gar nicht!) empfunden und damit Unzufriedenheit und Zoff zwischen Teammitgliedern entstehen. Diese Idee leitet sich offenbar vom Mythos ab, dass Teams sich selbst organisieren und das ein ganzes Team Verantwortung als Team zu tragen hat.

Die Prinzipien von Cockburn

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

Prinzip 1: Interaktive, persönliche Kommunikation ist die billigste und schnellste Art des Informationsaustausches. Dies besagt, dass Leute mit direktem Kontakt es einfacher haben, Software zu entwickeln, und die Software wird billiger sein. Wenn die Projektgröße zunimmt, ist natürlich der direkte Kontakt nicht mehr zu allen möglich, außerdem nimmt die Anzahl der Kommunikationswege überproportional mit der Anzahl Leute zu.
Mit der Projektgröße nehmen also auch die Kosten der Kommunikation zu und die Schwierigkeit der Software-Entwicklung steigt.
Anders ausgedrückt, die Effektivität einer Person nimmt ab, je mehr Leute beteiligt sind. Wenn also die Produktivität und Kosten im Vordergrund stehen und der Liefertermin egal ist, ist es vorteilhafter, kleine Teams zu haben.

[Anmerkung R.K.: So viel mündliche Kommunikation wie möglich, so genaue schriftliche Kommunikation wie möglich, beide wohldefiniert.]

Prinzip 2: Übergewichtige Methoden sind teuer.
Es soll nur das wirklich Nötigste produziert werden, und was darüber hinaus geht, ist mit Kosten verbunden. Wenn die Programmierer neben der wichtigsten Arbeit, dem Code zu schreiben, noch viel bürokratisches Material erzeugen müssen, werden sie zwangsläufig im Arbeitsfluss gebremst. Kleine Teams kommen mit wenigen Dokumenten zurecht, verwenden also eine leichtgewichtige Methode. Je größer aber ein Projekt ist, desto mehr Dokumentationen müssen geschrieben werden, umso schwerer wird die verwendete Methode. Deshalb ist es einleuchtend, dass kleinere Teams effizienter arbeiten können.
Die Methode kann aber auch zu leichtgewichtig werden. Dann kann die Qualität des Codes darunter leiden, oder die Programmierer sind nur noch damit beschäftigt, alten Code zu korrigieren.

[Anmerkung R.K.: Verwaltung minimieren. Genauigkeit des Codierten maximieren. Größe der Teams minimieren.]

Prinzip 3: Größere Teams brauchen schwerere Methoden.
Ein Team von bis zu zehn Leuten ist noch gut überschaubar. Werden es aber 30 Leute oder mehr, sind schon besondere Maßnahmen der Koordination nötig, man weiß nicht mehr genau woran die Anderen gerade arbeiten, es muss darauf geachtet werden, dass sich die Arbeiten nicht überlappen oder in die Quere kommen.
Kommunikation und Koordination sind also umso wichtiger, je größer das Team ist.

[Anmerkung R.K.: Kommunikation und Koordination sind sehr wichtig.]

Prinzip 4: (Nicht zutreffend)

Prinzip 5: Je mehr Feedback und Kommunikation, desto weniger Dokumentationen sind nötig.
Durch inkrementelle Entwicklung kann man laufend kleine, lauffähige Teile der Gesamtsoftware liefern. Der Auftraggeber kann sich darauf mit den Spezialisten für Anforderungsspezifikationen zusammensetzen und Korrekturen vornehmen. Dadurch entfallen ständige Anforderungs-Reviews innerhalb des Teams. Die Auswirkungen einer vorausgegangenen Tätigkeit können schneller erkannt werden, und dadurch verringern sich ständige Nachkorrekturen.

[Anmerkung R.K.: Klingt nach durchdachten Prozessen.]

Prinzip 6: Disziplin, Fähigkeiten und Wissen versus Prozesse, Formalismus und Dokumentation.
Jim Highsmith hat es so ausgedrückt: Verwechsle nicht Dokumentation mit Wissen. Was er meint, ist, dass sehr viel mit einem Projekt verbundenen Wissen nur in den Köpfen der Leute existiert und nicht auf Papier. Auch mit der besten Dokumentation kann ein neues Team nicht dort anknüpfen wo das alte aufgehört hat.
Highsmith weiter: Prozess ist nicht Disziplin. Disziplin bedeutet, eine Person wählt von sich aus eine bestimmte Art zu arbeiten, Prozess ist, die Person zu einer bestimmte Arbeitsweise zu zwingen. Natürlich hat das erstere davon einen besseren Einfluss auf ein Projekt.
Die dritte Unterscheidung von Highmith ist: Verwechsle nicht Formalismus mit Fähigkeiten. Gemeint ist, dass eine Person mit überragenden Fähigkeiten nicht ersetzt werden kann durch eine Person mit mittelmäßigen Fähigkeiten, die ihre Arbeit mittels Formalitäten und Anleitungen erledigt.

[Anmerkung R.K.: Der Punkt ist; Dokumentation ist nicht wissen, sondern Anleitung und Anweisung. Also, mit anderen Wörter, Dokumentation in den Händen eines Mitarbeiters erzeugt **nicht** Wissen (das Wissen muss schon vom Mitarbeiter erarbeitet werden).]

[Anmerkung R.K.: "Disziplin bedeutet, eine Person wählt von sich aus eine bestimmte Art zu arbeiten". Gut für die Person, wahrlich. Wenn aber alle "diszipliniert" (also so wie sie es wollen) arbeiten, ist mir eher schleierhaft, wie genau das gut fürs Projekt sein soll. Außerdem, es ist zwar verständlich, dass "Prozess nicht Disziplin" ist, doch Prozesse können in diesem (Projekt) Zusammenhang besser als "Disziplin" sein. Ein Beispiel? Gerne: die Programmier-Richtlinien.]

Prinzip 7: Effizienz ist erweiterbar in Aktivitäten, die nicht den Flaschenhals eines Prozesses darstellen.
Angenommen, in einem Projekt sind fünf Leute mit dem Programmieren von

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

[Anmerkung R.K.: Hört sich wichtig an, trägt aber keine neuen Erkenntnisse bei. Ich überlege, diesen Punkt auch als "nicht zutreffend" zu kennzeichnen.]

Funktionierende Teams: Eine einzige technische Leitung mit voller Autorität Entscheidungen zu treffen, auf der nächsten Stufe sind Assistenten, dann das dazugehörige technische Personal und zuletzt eine nicht-technische Person zum Unterstützten. Die Leistung des Teams wird dann durch den Leiter des Teams festgelegt. Die Größe des Teams und die Projektkomplexität ergibt sich aus der Fähigkeit des Leiters, das Problem zu verstehen und die Aufgaben zu verteilen. Diese Struktur vermeidet u.a. einen von Kreisel herbeigeführten Stillstand, erzeugt Verantwortung für die Aufgaben, die Größe des Teams kann optimiert werden. (Siehe hierzu **OP-Team**).

Anforderungen an einer Lösung für die SW-Entwicklung

Zunächst muss ich auf alle "Mythen" verzichten (Siehe **Mythen**). D.h., dass keine Mythos-Aussage ein Teil der Lösung sein darf. Zum Beispiel, sollte man behaupten, dass die Lösung nur funktioniert wenn sich Teams selbst organisieren, so ist diese Lösung zu verwerfen.

Das Design soll so weit flexibel sein, dass es schnell und einfach anzupassen ist.

Die Planung soll so weit flexibel sein, dass sie schnell und einfach anzupassen ist.

Die Lösung

1. soll einen einfachen Umgang mit Unberechenbarkeit und Komplexität gewährleisten.

2. soll Verschwendung minimieren, schlank und einfach sein.
3. soll stabil und reproduzierbar sein.
4. soll einen sogenannten "Plan B" nicht nur erlauben sondern regelrecht fördern.
5. braucht ein Regelwerk bzw. feste Richtlinien (diese sind von Organisation zu Organisation, Projekt zu Projekt, anzupassen). Siehe **Regelwerk**.
6. muss funktionieren bei Gruppen jeder Größe. Eine Lösung, die z.B. nur auf Gruppen von 4 bis 16 Mitarbeiter angewandt werden darf, ist als Lösung zu verwerfen.
7. soll immer anwendbar sein, unabhängig von der Organisation der Teams; Wasserfall oder Scrum oder was auch immer.
8. muss mit (sehr) großer Ungewissheit fertig werden. Sie muss so konzipiert sein, dass sie Ungewissheit meistert.
9. soll das Lernen von den Mitarbeitern fördern.

Über Entscheidungs-Meetings; der Leiter (Verantwortlicher) soll die involvierten und die, die über das Thema Ahnung haben, mit einbeziehen. Die Entscheidung wird aber von nur einer Person, der Verantwortlicher, getroffen. Das aber mit einer Begründung, die allen zugänglich sein soll.

Definieren was transparent sein soll. Unter anderem soll transparent sein wo jeder Mitarbeiter steht, was er macht, was ich von ihm erwarte.

Projektarbeit ist zwar immer neu und sich schnell ändernd, jedoch soll ich versuchen, so weit wie möglich Arbeiten wiederholen zu können, wiederkehrende Prozesse zu gestalten, Routine im Projektalltag einzubringen.

Die Entwickler sollen nicht motiviert werden, sondern gelobt für die gut entwickelte Software. Dafür sind sie da, sie wollen nicht mehr und nicht weniger.

Hindernisse (Impediments) sollen gleich angegangen werden; ein Bequatschen im Daily ist schon zu spät. Das Bequatschen ist an sich schon Verschwendung.

Problembehandlung: Probleme müssen in Erscheinung treten, und zwar so früh wie möglich, und sie müssen auch als Problem erkannt und bearbeitet werden. Das 'Management', so wie ich das kenne und kennenlernen durfte, will keine Probleme, sondern dass alles glatt läuft; das führt jedoch schnell zum Ignorieren von Problemen, somit lebt man (gern) in einer "Wahnvorstellung" der Realität (Wunschdenken). Das Management soll vielmehr dafür Sorge tragen, dass eine Geisteshaltung entsteht die Probleme sucht, erkennt und

behandelt; nicht unterm Teppich kehrt.

So früh wie möglich testen, verifizieren und validieren.

Das Fixen von Fehlern soll so schnell wie möglich erfolgen.

Mehr Qualität einbauen und weniger testen: je mehr Arbeit im Konzept fließt, desto weniger Fehler werden im fertigen Produkt zu finden sein.

"Gut genug" ist nicht gut genug.

Die Qualität muss im Rahmen der Lösung (im Prozess) eingebaut sein.

Man soll nicht damit rechnen, dass der Kunde immer unmittelbar zur Verfügung steht: Es soll folglich möglich sein, so weit wie möglich auf dem Kunden zu verzichten. Das ist die Realität; Der Kunde hat auch nicht ewig Zeit um sie in Meetings zu verbraten oder in kreativen Workshops oder was auch immer gerade im "Trend" liegt.

So viele Wörter wie möglich auf Deutsch; englisch soll nicht ein Teil der Lösung bzw. eine Quelle von "Schlagwörter" sein, welche die Wirklichkeit umnebeln und mangels Inhalt Unklarheiten erzeugen.

Wenn es erfahrungsgemäß gar nicht so einfach ist, ein Team zu bilden (geschweige denn eines, das sich selbst organisiert), ist vielleicht doch besser eine Lösung zu haben, die
1) keinem solch speziellen Team braucht und
2) die Interaktionen zwischen Mitglieder des Teams so einfach gestaltet, dass das Team quasi von alleine entsteht.

Eine Lösung soll dem Mitarbeiter entgegenkommen und Home-Office erlauben (wenn der Mitarbeiter krank ist, oder das Kind, Schwangerschaft, Krankheit oder Pflegefall in der Familie etc.).
Gründe:

1. Die Arbeit zu Hause ist produktiver: es gibt keine Unterbrechungen und keine Meetings
2. Ein verantwortlicher Projektmitarbeiter ist zu wichtig um auf ihn zu verzichten wenn er zu Hause bleiben muss.

3. Die Lösung soll auf (einzelne) Menschen setzen, nicht auf Teams, Prozessen oder Methoden.

Meiner Meinung nach gibt es kaum Gründe, die eine physische Anwesenheit eines Mitarbeiters auf der Arbeit bedingt.

Die Japaner sollen gemeint haben, "das einzige gefährliche Wettbewerb auf der Welt die Deutschen sind, wenn sie endlich gelernt haben, miteinander zu sprechen".

[Anmerkung R.K.: Das ist die japanische Sicht, die ich zwar verstehe, bin aber nicht überzeugt, dass sie stimmt. Das echte Problem bei den Deutschen ist die Verschwendung von Ressourcen (die sehen die Japaner nicht, oder wollen sie nicht sehen).
Nach alldem was ich gesehen habe, arbeiten die Deutschen und produzieren eine Menge, im Gegensatz zu anderen Länder, die nicht viel arbeiten und auch nicht viel produzieren. Doch der Deutsche arbeitet überflüssigerweise überfleißig: zu viel Anstrengung für einen bescheidenen Ertrag.]

Meetings als Zeitkiller sind zu minimieren.

Meetings sind giftig

Aus dem Buch: "Getting Real", von „37signals"

Vermeiden Sie Meetings!
Brauchen Sie wirklich ein Meeting? Meetings braucht man oft, wenn ein Konzept nicht klar genug ist. Anstatt zu einem Meeting aufzurufen, versuchen Sie das Konzept zu überarbeiten und zu vereinfachen, und senden Sie es schnell per E-Mail zur Diskussion. Das Ziel ist es, Meetings zu vermeiden. Jede Minute die man an Meetings sparen kann ist eine Minute an der man echte Arbeit verrichten kann.
Es gibt nichts giftigeres für die Produktivität als ein Meeting.

Hier ein paar Gründe, warum:

- Meetings zerstückeln Ihren Arbeitstag in kleine, nicht zusammenhängenden Teilchen, die Ihren natürlichen Arbeitsablauf stören
- In der Regel handeln Meetings mit Wörter und abstrakten Begriffe, nicht mit echten Entitäten (wie ein Stück Code oder ein Interface-Design)
- Meetings vermitteln in der Regel eine infinitesimale Menge an Informationen pro Minute
- Bei Meetings trifft man unweigerlich auf Wichtigtuer, die gerne die Arbeitszeit der Anderen mit unsinniges Gelaber verplempern

- Zu leicht driften Meetings vom eigentlichen Thema ab
- Oft haben Meetings sogenannte "Agendas", die so unscharf sind, dass kaum einer weiß, worum es wirklich gehen soll
- Meetings bedürfen einer gründlichen Vorbereitung, die kaum Jemand bereit ist zu tun.

Lösungsansätze

Vision: Zu wissen, was Sie erstellen wollen und warum.

Klarheit: Zu wissen, was Sie nicht erstellen wollen.

Focus: Zugang zu dem Endnutzer haben, dieser soll Sie in ihrer Arbeit so weit einführen, dass Sie sie verstanden haben.

Die Erfolgreiche wissen das: Niemand kann nicht genug vorbereiten. Folglich kann man auch nicht genug planen. Die Denkarbeit ist zwar anstrengend, doch sie "kostet" bei Weitem weniger als die "Arbeit" zu verrichten (z.B. programmieren). Daher ist eine gute Planung im wahrsten Sinne des Wortes unbezahlbar. Es ist klar, **es ist von Anfang an klar**, dass die Planung ab einem gewissen Zeitpunkt die Wirklichkeit nicht mehr entsprechen wird, doch das ist kein Grund um nicht zu planen (höchstens eine Ausrede für Ultrafaule). Jeder Gedanke, den ich mich im Vorfeld gemacht habe, kommt mir zu Gute wenn er gebraucht wird: Also; "genug planen" ist nicht möglich. Klar?
Das Gleiche gilt für Design, Architektur, Systemanalyse, SW-Engineering, Organisation, Qualitätssicherung etc. Die Tätigkeiten diese Experten müssen durchaus im Laufe des Projektes weiter stattfinden.

Das Gedanken-Machen hilft ungemein, mögliche Änderungen (auch später im Laufe des Projektes) aufzunehmen und einzubauen. Aus diesem Grund sind gute und erfahrene Designer und Planer aus unschätzbarem Wert.

Gute Designer und Systemanalytiker sind wichtig, denn die Lösung soll flexibel sein, um sich die kommenden Änderungen anzupassen.

Projektmanagement durch "hingehen und anschauen" (Projekt Managing by walking around) – Genchi-Genbutsu auf japanisch.

Ein Wasserfall existiert in der Form nicht; es handelt sich um eine Kette. Eine Verkettung von Prozesse, Ereignisse oder Produktfertigungsphasen. Eine Kette von Phasen.

Scrum fokussiert zu stark auf Prozesse. Die Produktion wird außer Acht gelassen. Teams sollen sich selbst organisieren. Mitarbeiter werden außer Acht gelassen.

Ein bisschen Menschenkenntnisse bei den Mitarbeiter: Jeder Stakeholder ist in Wirklichkeit stets bestrebt, folgende persönliche Strategie durchzuführen;

- eigene Verantwortung zu minimieren: Bequemlichkeit
- Arbeitsvolumen zu minimieren: Wahrscheinlichkeit von eigenen Fehlern minimieren
- Arbeitseffektivität zu minimieren: Das Projekt soll so lange wie möglich andauern (Einnahmen maximieren, kein Stress mit neuem Job etc.).

Daher sind Wege zu finden um die Ziele der Mitarbeiter mit den Zielen des Projektes im Einklang zu bringen.

Zu berücksichtigen bezüglich Mitarbeiter; Sie sind Menschen.
Menschen sind faul, bequem und egozentrisch. Weiterhin sind sie auch: neugierig, experimentierfreudig, wollen Spaß haben und sich wichtig fühlen. Außerdem mögen Menschen Herausforderungen, besser werden, Neues lernen und Etwas produzieren.
Folglich muss eine Lösung alle diesen Eigenschaften Rechnung tragen, nicht dagegen kämpfen (mit erniedrigenden Daylies, infantilen Retrospektiven, kindlichem Planning-Poker, verblödenden User Stories: "As an xxx I want a yyy", unsinnigen Story Points etc.).

Verantwortungsübernahme bei den Mitarbeiter fördern.

Zwiebel-Prinzip: Organisatorisch gesehen ein Entwickler-Kern und die anderen Mitarbeiter drum herum.

Auftragsabwicklung nach den Grundsätzen der inneren Führung (Bundeswehr).

Nur so viele Mitarbeiter wie Aufgaben zu erfüllen sind: Schlanke Organisation.
Die Menschen dezidiert einsetzen.

Es wäre wünschenswert, wenn die Entwickler eine "Kernzeit" hätten, wo sie in Ruhe und ohne Unterbrechungen entwickeln könnten.

Ein Team braucht einen Auftrag, sonst sind es nur zusammengewürfelte Menschen ohne Ziel.

Projektplanung, Software Engineering und SW-Entwicklung als Analogie zur Sprache. Folgendermaßen:

- Hauptwörter (Produkte)
- Verben (Aktivitäten)
- Grammatik (Regelwerk bzw. Regeln zwischen die Produkte und die Aktivitäten).

Ein passender Regelwerk:
- So viel Regel wie nötig, so wenig wie möglich.
- Regeln sollten ständig angepasst werden.
Siehe **Regelwerk**.

Zu beachten auch bei einer Lösung

Aus Erfahrung ist bekannt, dass die Kunden oft schnell noch eine Änderung schieben wollen, weil sie fühlen, dass je schneller eine Änderung in die Wege geleitet wird, desto einfacher die Implementierung sein wird. Nun ist es auch oft so, dass diese Änderung doch nicht so wichtig war wie der Kunde zuerst angenommen hatte. Deswegen macht es Sinn, eine "tote Zeit" zu installieren zwischen Änderungswunsch und die tatsächliche Durchführung der Änderung.

Regelmäßige Lieferungen (wie Scrum fördert nach jedem Sprint) machen Sinn: erwecken Vertrauen, bestätigen, dass die Entwicklung richtig läuft und hilft den Kunden, die Anforderungen ggf. anzupassen. Zu beachten ist lediglich, dass die Vorstellung von fixen Sprints mit lieferbaren Ergebnisse schlicht realitätsfremd ist. Folglich sollen die „Lieferungen“ mit Vernunft geplant werden

Fazit

Ich bin überzeugt, dass Wissen und Können von Menschen die SW entstehen lässt, weder Entwicklungsmethoden noch "5GL" Programmiersprachen noch

„Parametrisierbare SW“ noch sich selbst organisierende Teams noch übermotivierte Mitarbeiter die Arbeit verrichten, sondern die richtige Mischung von alledem in der richtigen Menge bringt uns zum gewünschten Ergebnis.

Und, als Qualitätssicherer nehme ich mir die Freiheit, hier die Wahrheit, so wie ich sie sehe, auszusprechen. Echte Qualitätssicherer müssten das leider so machen: Wunde Punkten sind das Metier des Qualitätssicherers.

Das heutige "agil" ist zu ca. 75% Verschwendung.

Man kann nicht "agil" sein, wenn man in einem Korsett von Meetings, Planings, Retros, Groomings etc. agieren muss.

Die erwünschte und als Ziel erkannte "Agilität" muss im Kopfe der Mitarbeiter entstehen, und außerdem, und das hier ist der Kasus Knaktus, muss vom Prozess in den Organisationen "erlaubt" werden, nicht "erzwungen".
Das ist der Weg, so wie ich ihn sehe. Ist das aber möglich? Und zu welchem 'Preis'? Macht es überhaupt Sinn, Anstrengungen in dieser Richtung zu starten? Ich bin mir nicht wirklich sicher. Wenn ich den sogenannten agilen Ansatz entmystifiziere (also von allen den oben ausgeführten Mythen befreie): Was bleibt über? Ist dieses Gerippe noch zu gebrauchen? Ich hege sehr ernste Zweifel.

Aus meiner Sicht leiden die Unternehmensleitungen bzw. Geschäftsführungen im Allgemeinen unter den negativen Einfluss der angelsächsischen Philosophie (Mentalität) der ungesunden (Über-)Kontrolle, starren Hierarchie, Oben-versus-Unten-Schicht-Mentalität. Für Deutschland und die Deutschen passt diese Mentalität nicht; sie ist nicht "gesund", nicht für alle Teilnehmer zufriedenstellend: Jeder Mitarbeiter in Deutschland wird immer sein bestens aus Überzeugung tun, was eigentlich bei den überkontrollierten Unterschichten der angelsächsischen Gesellschaft nicht der Fall ist.

Ich will nicht gegen die "Agilisten" in einem Kampf ziehen; sie dürfen machen, was sie für richtig halten. Ist auch nicht gesagt, dass in bestimmten Konstellationen so ein agiler Ansatz nicht funktionieren kann, ...vielmehr scheint die Erfahrung zu zeigen, dass der Ansatz doch funktioniert, obgleich sicherlich nicht optimal (die Entwickler bringen nicht das Ergebnis, was sie bringen könnten).

Wenn meine Ausführungen hier aber böse Kommentare und Kritik ernten, so

lasse ich dies als Beweis von einer fast fanatischen Ideologie gelten, von fundamentalistischem (dogmatischem, in religiösem Sinne) Glauben bei so manchem "Agilist".

Mein Anliegen ist halt, die Projektarbeit und die SW-Entwicklung zu optimieren und beide weiter zu entwickeln.

Und das mit einem geänderten Ansatz aus Lean und aus Projekterfahrungen: **Flexibles Planen und Arbeiten** (FlePA).

FlePA und "Agile"

Die von mir erstellten Lösung berücksichtigt so gut es ging alle Punkte der Anforderungsliste und adressiert die angesprochenen Probleme der SW-Entwicklung.

FlePA stellt ein Rahmen zur SW-Entwicklung zur Verfügung, diesen Rahmen können Sie jedoch auch auf andere Arten der Projektarbeit (nicht nur SW-Entwicklung) anwenden. FlePA ist schlank gehalten, dergestalt, dass es nicht dafür gedacht ist, Portfolio- oder Programmmanagement zu machen, sondern findet sich wieder auf der Ausführungsebene von Projekten.

Schwerpunkte von FlePA sind:
1. Flexible Planung
2. Flexibles Design
3. Eine Kommunikation, die stark in den Arbeitsprozessen eingreift.

Leider werden die "Agilisten" mit dieser Lösung, meiner Einschätzung nach, ziemlich unglücklich werden, denn der agile Ansatz, vom Manifest über XP bis zu Kanban, wurde magerer und magerer so wie ich mit den Arbeiten voranschritt.

Dennoch denke ich, dass ich bekannt machen sollte was aus "Agile" wurde:

1. Der Punkt (Begriff) "Agile" habe ich auf "Berücksichtigung von menschlichen Bedürfnissen" umbenannt

2. Der Ansatz von "Agile" konnte ich auf 3 Punkte komprimieren:
   - Arbeitsmenge minimieren
   - Verantwortlichkeiten minimieren
   - Arbeitseffektivität minimieren

3. Der "agile" Ansatz erlaubt bei mir immerhin drei Aktivitäten:
   - Entscheidungen treffen
   - Experimentieren
   - Fehler machen (lernen)

Trotz (oder genauer gesagt mit Hilfe von) dem 2. und dem 3. Punkt dieser Liste (die Berücksichtigung von menschlichen Bedürfnissen) ist der FlePA Rahmen in der Lage die Entwicklung zum "fließen" zu bringen und zielorientiert ein Ergebnis liefern.

Hintergründe dieser Untersuchung

Im Grunde genommen habe ich, als Techniker, Mathematiker, Systemanalytiker und Qualitätssicherer, unter Berücksichtigung auch der vielen Jahre Erfahrung die ich mit mir schleppe, sehr ernste Bedenken, dass in Projekten sich Arbeitsergebnisse auf wundersamer Weise einstellen, nur weil das z.B. im Dogma der agilen Methoden so stillschweigend hingenommen wird.

Es ist sehr Schade, wenn man weitgehend auf kompetente und nützliche Leute wie z.B. Designer, SW-Architekten, Analysten, Software Engineers etc. verzichtet, nur weil sie im agilen Manifest-Pamphlet ignoriert wurden.

Darüber hinaus finde ich es auch Schade, dass man Qualitätsmanager (also Leute mit breitem Wissen über das Thema Qualität) als Tester "missbraucht".

Und nein, es ist nicht eine Kampagne gegen die agilen Methoden: Ich habe lange Zeit selber versucht die Vorteile der Agilen zu finden, bevor ich aufgegeben habe. Wenn Vorteile bei den Agilen zu finden sind, bin von den Ersten, diese zu würdigen. „Lean“ und „Kanban“ sind z.B. Ansätze, die meiner Meinung nach nicht sinnentleert sind; sie sind zwar für die Serienproduktion entwickelt, man kann sie aber dennoch („suboptimal“, wie man so schön sagt) auch für die Projektarbeit anscheinend einsetzen. Finde ich gut, nicht optimal, aber „gut genug“ allemal.

Anstelle eines Epilogs

(Diese Geschichte ist nicht von mir!)

Diese Geschichte ist 100% wahr.

Ich habe einen Auftrag bekommen um mit einem Team von Bauarbeitern ein Haus zu bauen. Wir hatten ein Meeting mit dem Management, um herauszufinden, was für ein Haus sie haben wollten. Zeichnungen, Grundriss, wofür war das Haus gedacht, wie viele Leute sollten hier wohnen, usw. Das Management sagte, dass sie das noch nicht wüssten, doch wir sollten schon einfach loslegen. Sie verrieten uns, dass wir "agil" bauen sollten und das bedeuten würde, dass mit kleinen Lieferungen nach und nach, "schrittweise" (mit der Bedeutung; einen Elefanten einem Häppchen nach dem anderen aufessend) das Haus zu bauen wäre.

Das Entwicklerteam meinte hierzu, dass man zumindest wissen müsste wie groß das Haus werden sollte, so dass man den Fundament gießen kann. Daraufhin meinte das Management, dass sie es schlicht noch nicht wüssten. "Das einzige, was wir wissen, ist dass wir eine Toilette haben möchten" war die verbindliche Aussage, und wir sollten einfach schon damit starten. In 2 Wochen sollte sie fertig sein und dann würde das Management auch etwas mehr über das Vorhaben wissen.

Also kauften wir einen Kübelklo und stellten es hin. In 2 Wochen wurde es sehr hübsch, mit einer Malerei und fein geschnitzte Fenster. Doch das Management war sichtlich unzufrieden: "Ist das ein Haus? Das ist das schlechteste Haus, dass wir je gesehen haben! Nicht mal ein Fernseher ist drin!".

Also haben wir einen Fernseher gekauft und in einer Ecke des Kübelsklos angeschraubt. Der Strom für das Gerät wurde durch einen Motogenerator neben dem Kübelklo erzeugt. Wir wollten schon einen DVD-Player kaufen, doch das Management lehnte diese Ausgabe mit der Begründung ab, dass sie sich nicht sicher waren, ob sie das überhaupt haben wollten. Sie meinten, sie würden ggf. darauf zurückkommen wenn es aktuell werden würde.

Das Management entschied nun, dass wir auf jeden Fall etwas Stauraum benötigen, so kauften wir eine Auto-Ladefläche und klebten sie mit Panzerklebeband am Kübelklo an. Aber: Oh, Schreck! Das Management lies einige wenigen Angestellten an jeweils kleinen Tischchen dort arbeiten! Und sie (das Management) mochten das aber wirklich...

Nach ein paar Jahren wuchs die Ladefläche zu einem großen, baufälligen Komplex. Regelmäßig gab es Überschwemmungen, bei Regen leckte es hier und dort, im Winter war sie eiskalt und in Sommer wie ein Ofen. Rein und Raus musste man durch ein Labyrinth von Röhren gehen, hängenden Kabeln und einfach so hingestellten Gegenständen, die Niemandem gehörten. Da die erste Ladefläche so gut funktioniert hatte (aus Sicht des Management) wurde auch eine zweite angeschafft. Der alte Motogenerator wurde natürlich für alle Gebäuden gebraucht, was er nicht schaffte, daher musste der Verbrauch

rationalisiert werden.

Die Kommunikation zwischen den Gebäuden war bald ein Problem. Für das eine von beiden verwendeten wir ein komplexes System von Flaggensignale. Für das andere Gebäude wurden von den Mitarbeitern Notizen auf Papier geschrieben, zerknittert und rüber geworfen. Beide Methoden wurden zwar als Witze vorgeschlagen, doch das Management war aus irgendeinem Grund wirklich begeistert. Die Mitarbeiter in den Gebäuden müssen eigentlich miteinander kommunizieren, doch das müssen sie durch unsere Abteilung tun, folglich hatten wir die meiste Zeit die Aufgabe, Nachrichten weiterzureichen.

Es wurde auch vorgeschlagen, dass wir Flugzeuge aus Papier anstatt der zerknüllten Papierkugeln verwenden sollten, doch die breiten, ungeschickten Finger des Geschäftsführers, der diese Kommunikationswege zu benutzen pflegte, waren nicht imstande das Papier richtig zu falten; auch nicht nach mehreren Trainingskurse. Ich persönlich habe eine Papierflugzeugfaltmaschine gebaut. Doch wiederum konnte der Geschäftsführer nicht dazu trainiert werden, diese Maschine zu bedienen. Also haben nach wie vor die zerknitterten Papierbälle ihr Dienst geschoben.

Aber das Schlimmste von dieser Geschichte ist, dass alles in Betrieb ist. Alle Mitarbeiter sind unglücklich, aber das Unternehmen verdient Geld mit dem Ganzen. Das Gute aber ist es, dass die Arbeiten weitgehend abgeschlossen sind, und nun wissen wir, was der Kunde wirklich haben wollte. Also können wir damit anfangen, das Projekt neu auszurollen. Klar, dass wir dem Management von unserem Vorhaben nichts sagen dürfen solange es nicht fertig ist. Sie haben das riesige Loch im Boden, wo das Fundament kommen soll, schon gesehen, doch wir haben angegeben, dass es sich um einen Lagerplatz für Back-Ups handelt. Daraufhin ignorierte das Management geflissentlich das Loch, solange kein Kunde reinfiel.

Wahrscheinlich werde ich selbst nicht mehr hier sein wenn das neue Gebäude fertig wird; ich kann 50% mehr Gehalt kriegen wenn ich den Job wechsle. Immerhin meint unser Management hier, dass ich keine Gehaltserhöhung verdient habe weil ich einige Sprints verpasst habe.

[Anmerkung R.K.: Vertraut an dieser Geschichte kam mir durchaus die kindliche Begeisterung des 'Managements' für Maßnahmen vor, die offensichtlich von vornherein eine Missgeburt waren. An 2. Stelle der Vertrautheit rangierte das Ignorieren von den Missständen, die sie nicht sehen wollen.]

Abkürzungen

... LOC : ... Lines of Code
ALM : Application Lifecycle Management
ATDD : Acceptance Test-Driven Development
BDD : Behavior-Driven Development
BDUF : Big Design Up Front
CD : Continuous Delivery
CI : Continuous Integration
CI / CD : Continuous Integration / Continuous Delivery
CIP : Continuous Improvement Process
CoP : Community of Practice
DI : Dependency Injection (Pattern)
DRY : Don't Repeat Yourself
IoC : Inversion of Control (Pattern)
JEDUF : Just Enough Design Up Front
LAMDA : Look-Ask-Model-Discuss-Act
LSD : Lean Software Development
MINT : Mathematik, Informatik, Naturwissenschaften, Technik
MVVG : Mission, Vision, Values, and Goals
MoSCoW : Priorisierung (Must Have / Should Have / Could Have / Won’t Have this Time)
OOP : Object Oriented Programming
PIL : Product Innovation Lifecycle
RAD : Rapid Application Development
RFP : Request for Proposal
TDD : Test-Driven Development
TPS : Toyota Production System
XP : Extreme Programming
YAGNI : You Aren't Going to Need It (oder eher "You Are Going to Need It"?).
Besser:
YWNNI : Youw will not need it

Links

Hier einige Links. Auf gar keinen Fall sollten Sie aber meinen, der eine oder andere "Recht" oder "Unrecht" hat: Jede Situation, jedes Projekt, jedes Team ist anders und von daher ist eine absolute und allgemeingültige Wahrheit nicht zu finden.

http://blog.toolshed.com/2015/05/the-failure-of-agile.html

http://darkagilemanifesto.org

http://gilesbowkett.blogspot.de/2014/09/why-scrum-should-basically-just-die-in.html

https://ggreiter.wordpress.com/2016/01/28/uber-falsch-verstandene-agile-methodik/

http://intrinsify.me/Blog/items/so-funktioniert-fuehrung-ohne-vorgesetzte.html

http://intrinsify.me/Blog/items/wie-moderne-unternehmensfuehrung-ohne-helden-auskommt.html

http://joostdevblog.blogspot.de/2014/03/why-scrum-is-fundamentally-broken-but.html

http://lostechies.com/jimmybogard/2012/09/12/why-im-done-with-scrum/

http://ronjeffries.com/xprog/articles/beyond-agile-new-principles/

http://ruynk.blogspot.com

http://simpleprogrammer.com/2010/02/23/scrum-will-die/

http://typicalprogrammer.com/why-dont-software-development-methodologies-work/

http://venturebeat.com/2014/07/30/u-s-govt-heres-what-went-wrong-with-healthcare-gov/

http://www.cio.com/article/2380827/developer/developer-6-software-development-lessons-from-healthcare-gov-s-failed-launch.html

http://www.cio.com/article/2385322/agile-development/why-agile-isn-t-working--bringing-common-sense-to-agile-principles.html

http://www.ruynk.de

http://www.ryusui.de

http://www.status-quo-agile.de

https://age-of-product.com/engineers-despise-agile/

https://michaelochurch.wordpress.com/2015/06/06/why-agile-and-especially-scrum-are-terrible/

https://www.quora.com/In-a-nutshell-why-do-a-lot-of-developers-dislike-Agile