Der Agile Softwareentwickler

Will man von einem traditionellen (wohl Wasserfall orientierten-) Ansatz auf einen Agilen Softwareentwicklungs-Ansatz umstellen, so kommt man nicht um prominente Vertreter wie Scrum oder Kanban herum. Ehe man sich versieht, setzt man sich mit Rollenbeschreibungen wie dem Product Owner oder Scrum Master auseinandersetzen. Die Unmengen an Büchern, Kurse und natürlich Blog Posts helfen einem dabei. Was bei einer solchen “Agilen Transformation” meist ausser Acht gelassen wird ist, dass sich auch die Rolle des Entwicklers sehr stark ändert. Erkennt man dies nicht, droht die Transformation zu scheitern.

In diesem Artikel möchte ich meine Gedanken zur Rolle eines agilen Softwareentwicklers teilen.

## Wasserfall vs. Agil

In einem typisch Wasserfall orientiertem Vorgehen ist der Softwareentwickler primär in der Entwicklungs-Phase involviert. Im Gegensatz dazu ist er bei einem Agilen Vorgehen in allen Projektphasen vertreten. Natürlich bedeutet dies nicht, dass er auf einmal die Anforderungen für das Produkt selbst erhebt und damit die “Requirement Engineers” Arbeitslos macht. Es geht dabei vielmehr darum, dass bei einem agilen Vorgehen die verschiedenen Diszplinen sehr eng miteinander zusammenarbeiten und dadurch alle daran beteiligten Personen ihr Spektrum erweitern müssen.

Wasserfall vs. Agil

Wasserfall vs. Agil

Das Entwickler-Team wird also bereits bei der Anforderungserhebung miteinbezogen. Dies ist sehr entscheidend, da nur die Entwickler selbst das Know-How die Implementierung der Software haben. Sie sind die einzigen, welche beurteilen können wie eine neue Anforderung implementiert werden soll, ob es eine einfachere und güstigere Alternative für das Problem gibt, oder ob die Architektur für kommende Anforderungen gewappnet ist oder nicht.

Die Agile Vorgehensweise sagt meiner Meinung nämlich vorallem eines aus: **Diejenigen, die die Arbeit machen, entscheiden auch wie sie sie machen. Dies betrifft sowohl die technische Umsetzung, wie auch die Vorgehensweise, mit welcher die Software entwickelt wird.**

Im Folgenden will ich genauer auf die (meiner Meinung nach) relevantesten Eigenschaften und Fähigkeiten eines Entwicklers im Agilen Umfeld eingehen.

## Teamfähigkeit

Die Fähigkeit im Team arbeiten zu können muss jeder beteiligte einer agilen Umgebung besitzen, nicht nur der Entwickler. Sieht man sich das Agile Manifest [[1]](#footnote-22) genauer an, so sieht man, dass gleiche mehrere der dort definierten Werte und Prinzipien darauf abzielen:

**Individuen und Interaktionen** mehr als Prozesse und Werkzeuge **Zusammenarbeit mit** dem Kunden mehr als Vertragsverhandlung

sowie

Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.

### Was ist Teamwork?

Teamfähigkeit bzw. Teamwork wird leider sehr oft missverstanden. Vielleicht erkennt man sowas erst wenn man selbst einige Zeit in einem selbstorganisierten Team mitgearbeitet hat. Ich habe mich schwergetan, eine einheitliche Definition für Teamwork auszuarbeiten. Stattdessen habe ich einige Punkte aufgelistet, was für mich Teamwork beduetet, bzw. eben nicht:

* Teamwork bedeutet nicht, dass ein Teamleiter die Aufgaben an Teammitglieder vergibt.
* Teamwork bedeutet nicht, dass nur ein Mitglied das Wissen über eine Aufgabe hat.
  + Das Team sollte die Aufgabenverteilung selbst vornehmen und zumindest dabei sein. Somit sind die Aufgaben auch allen Bekannt und eine Aufgabe ist nicht mehr nur einem Mitglied zugeordnet. Ansonsten ist der Sinne eines Teams meiner Meinung nach nicht gewährleistet. Teamwork beschreibt ja gerade, dass Aufgaben vom Team und nicht von einzelnen Mitgliedern ausgeführt werden.
  + Ein guter Anhaltspunkt ist die Ersetzbarkeit eines einzelnen Mitgliedes. Ein Team ist in der Lage, eine gewisse Zeit auf einzelne Mitglieder zu verzichten.
* Teamwork beudeutet, dass Probleme im Team besprochen und gelöst werden. Tritt also bei einer Aufgabe ein Problem auf, so wird dieses im Team diskutiert und die Lösung, welche für das Team am besten passt wird ausgewählt. Auch hier werden wieder alle Mitglieder über das Problem informiert.
  + Auch dieser Punkt geht in die Esetzbarkeit eines jeden einzelnen hinein.
  + Werden Lösungen für Probleme von einzelnen Mitgliedern getroffen, so hat dies mehrere Nachteile. Einersetzs ist die Ersetzbarkeit nicht gewährleistet, solange das Problem noch nicht vollständig gelöst ist. Andererseits lernt nur ein Mitglied von diesem Problem. Probleme sind ähnlich wie Fehler - man sollte von ihnen lernen.
* Teamwork beudetet, dass alle Mitglieder miteinander und füreinander arbeiten. Die Arbeit ist erst getan wenn das ganze Team fertig ist. Die Leistungen der Individuen hat keine Relevanz, es zählt ausschieslich die Teamleistung.
  + Teamwork beduetet nicht, dass Egebnis aufgrund einer Person gut oder schlecht ist. Es ist immer das ganze Team verantwortlich für Erfolge und Niederlagen.
* Teamwork bedeutet, dass es keine Hierarchien gibt.
  + Keine Hierarchien impliziert nicht, dass auch keine Strukturen vorhanden sind.
  + In einem guten Team gibt es erfahrene wie auch weniger erfahrene Mitglieder. Anstatt dass Erfahrene Mitglieder hierarchisch hervorgehoben werden, sind in einem Team vielmehr Leader Qualitäten gefragt. Mitglieder mit weniger Erfahrung müssen die Qalitäten aufweissen sich das Wissen schnell von anderen Mitgliedern “abzusaugen”.
  + Teamwork beduetet nicht, alle Arbeit auf sich zu ziehen und diese dann zu delegieren. Ein oft beobachtetes Phänomen, wenn man neu in einem Team zusammenarbeiten soll. Falls man durch Mehrwissen oder mehr Erfahrungen gewisse Arbeiten auf sich zieht, muss man damit uzugehen wissen. Man muss als Leader agieren und andere Mitglieder dabei unterstützen diese Fähigkeiten ebenfalls zu erlernen. Man ist verpflichtet, seine eigene Kapazität im Griff zu haben.
  + Ein Leader hat nichts mit einem klassischen Teamleiter zu tun. Siehe auch Leader vs. Boss.
* Teamwork beduetet, dass man auch weniger erfreuliche Arbeiten zugunsten des Teams erledigt.
* Teamwork bedeutet, dass ständig eine Ansprechsperson vorhanden ist.
* Teamwork bedeutet, dass eine aktive Feedbackkultuir zwischen den Mitgliedern lebt. Jeder kann mit Kritik umgehen uns weiss diese zu verarbeiten. Jeder ist Einsichtig und nimmt Kritik entgegen. Jeder kann seine Meinung äussern und vertreten. Niemand ist rechthaberisch.
* Teamwork beduetet, dass Kritik immer konstruktiv plaziert wird.
  + Teamwork beduetet, dass man Froh um konstruktive Kritiv und immer offen dafür ist.
  + Es sollte ein familiäres Verhältnis zwischen den Teammitgliedern herschen. Jedes Teammitglied sollte sich sicher fühlen. Fehler sind normal und notwendig um zu lernen.
* Teamwork beduetet Ehrlichkeit in allen belangen. Dies führt zu Vertrauen.
* Teamwork bedeutet, dass man gewisse Arbeiten zusammen erledigt.
  + Gerade Programmierer neigen dazu, alles am eigenen Rechner machen zu wollen. Arbeitet man im Team, so werden gewisse arbeiten auch zusammen ausgeführt (Stichwort “Pair-Programming”). Ein Mitglied muss also auch mal mehrere Tage neben einem anderen Mitglied am selben Rechner verbringen können.

### Untypische Eigenschaft für Programmierer

Die Zeiten, als ein einzelner Programmierer das Know-How über die gesammte Software hat und diese alleine nach seinen Vorstellungen von Architektur und Design entwickelt ist vorbei. Dies ist nicht nur nicht Businessorientiert, es skaliert auch nicht und ist kein langfristiger Ansatz. Solche Software wird früher oder später neu geschrieben werden müssen, egal wie exzellent und sauber sie aufgebaut ist.

Vielleicht sind wir nicht deswegen in das Metier der Programmierer gelangt, um mit anderen zu kollaborieren. Tja, Überraschung! Beim Programmieren geht es vor allem um die Zusammenarbeit mit anderen. Wir müssen mit unserem Business zusammenarbeiten - genauso wie auch mit unseren Programmierkollegen. [[2]](#footnote-25)

Im Paper über Extreme Programming (XP) [[3]](#footnote-26) wurde dies ziemlich gut zusammengefasst: > Die Entwickler und Entwicklerinnen müssen in einem hohen Mass teamfähig sein, da gerade das Entwickeln in Paaren eine hohe Sozialkompetenz voraussetzt. Dies ist insbesondere wichtig, da in XP der Programmierer neben der eigentlichen Programmierung auch für die Aufwandschätzung und für das Erstellen der Tests verantwortlich ist, also Aktivitäten, die immer auch mit den anderen Personen koordiniert werden müssen. Einzelgänger, die nicht über ihre Arbeit diskutieren können oder nicht kritikfähig sind, eignen sich nicht für ein XP-Team. Der Teamerfolg muss in jedem Moment über dem eigenen Erfolg stehen. Ängste sich zu blamieren, veraltete Ansichten zu vertreteten, nutzlos zu werden oder den hohen Anforderungen nicht zu genügen, müssen überwunden werden, um erfolgreich in einem XP-Team mitarbeiten zu können. Dies ist gerade für die Programmierer, die in der Regel nicht besonders kommunikativ sind, eine besondere Herausforderung (vgl. Beck 2003: 141).

## Kommunikation

Die Anforderungen an die Kommunikationsskills eines Softwareentwicklers verändern sich bei einer Umstellung auf Agil enorm. Während ein Softwareentwickler im klassischen Wasserfall modell quasi “Stupid” die vorgegebenen Anforderungen und Designs implementiert muss er im Agilen Umfeld mit diversen anderen Rollen kommunizieren können. Ob Stakeholder, Auftraggeber, End-User, Product Owner, Scrum Master, Tester, UI-/UX Spezialisten oder andere Entwickler - er muss mit ihnen allen kommunizieren können. Viele dieser Rollen sprechen eine ganz andere Sprache und verstehen oft unter selbem Begriff etwas komplett anderes.

Vielleicht sind Sie “nur ein Programmierer”, aber die Fähigkeit, sich mit einem Geschäftskunden in der Sprache seine Branche zu unterhalten, ist für Ihren beruflichen Erfolg lebenswichtig. [[4]](#footnote-28)

Die Kommunikation mit anderen Rollen erfordert “Soft-Skills” wie Epathie. Es gehört Beispielsweise zu den Aufgaben eines Agilen Softwareentwicklern gewisse Anforderungen zu hinterfragen und nach dem wahren Problem dahinter zu Fragen. Dies kann von der Gegenseite oft falsch aufgefasst werden. Er muss seinen Standpunkt in gewissen Diskussionen klar vertreten können. Gerade im täglichen Krieg zwischen Business- und technischen Anforderungen muss er die technischen Eigenheiten erläutern und auf den Punkt bringen können. In grösseren Agilen Umgebungen kann dies zum Teil etwas politisch werden - auch diese Fähigkeit muss er beweisen und gewisse “lobying” Arbeit leisten.

Extreme Programming recognizes the importance of design decisions, but it strongly resists upfront design. Instead, it puts an admirable effort into communication and improving the project’s ability to change course rapidly. [[5]](#footnote-29)

When the customer says, “You must do this and this and this,” you have to be prepared to discuss which of those items is really necessary and how much of each. [[6]](#footnote-30)

## Technische Exzellenz

Dass ein Softwareentwickler in seiner Programmiersprache fit sein muss wird niemand bestreiten. Natürlich ist dies im Agilen Umfeld nicht anders. Jedoch kommen einige Aspekte hinzu. Da der Entwickler keine fest definierten Vorgaben hat sind plötzlich Kenntnisse und Fähigkeiten in Software- Design und Architektur Pflicht. Es gibt keinen fixen Plan den der Entwickler “nur noch” in Code umwandeln muss[[7]](#footnote-32). Im Agilen Manifest stehen ebenfalls einige wichtige Sätze zu dieser Thematik:

Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität. Einfachheit - die Kunst, die Menge nicht getaner Arbeit zu maximieren - ist essenziell. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams. [[8]](#footnote-33)

Ein Agiler Entwickler hat ein produktorientiertes Denken, ein traditioneller Entwickler eher ein Projektorientiertes. Damit meine ich, dass das Ziel eines Agilen Softwareentwicklers ist, eine tolle Lösung für das Problem des Anwenders zu entwickeln. Dabei legt er natürlich Wert auf Qualität, Wartbarkeit sowie Erweiterbarkeit, weil das Produkt und nicht das Projekt im Augenmek liegt.

*[..]* developers without solid design principles will produce a code base that is hard to understand or change - the oppisite of agility [[9]](#footnote-34)

Die Technische Exzellenz manifestiert sich jedoch nicht nur im Sourcecode. Es gehört zu den Aufgaben eines Entwicklers Aufwände sowie Risikos zu anfallenden Anforderungen abzuschätzen. Er muss technische Anforderungen vertreten, begründen und dafür werben können. Er muss sich zudem mit Prinzipien wie Continuous Integration sowie Deployment auskennen, da diese ein Enabler für kurze Feedback-Zyklen und wirkliche Agilität sind.

## Domänenwissen

Der Beruf Softwareentwickler trägt mit sich, dass man sich immer in zwei Fachbereichen auskennen muss. Der eine ist natürlich der IT- bzw. Software- Fachbereich. Dieser alleine gehört zu den komplexeren. Der jeweils andere ist der Fachbereich, wofür die Software-Lösung entwickelt wird. Da der agile Softwareentwickler sehr nahe mit Rollen des Requirement-Engineering zusammenarbeitet, muss er sich auch mehr in die Domäne einarbeiten, als wenn man die konkreten Anforderungen vorgesetzt bekommt. Dies erfordert den Willen sich in eine zum Teil fremde Domäne einzuarbeiten und diese versuchen zu verstehen.

Der Begriff “Domain-Driven Design” welcher vom gleichnamigen Buch von Eric Evans stammt impliziert genau dies. Nicht per Zufall wird bereits im Vorwort das agile Entwicklungsvorgehen betont und der Zusammenhang zwischen Domänengetriebenem Softwaredesign und agiler Vorgehensweise erläutert.

## Empirisches Mindset

Ein agiler Softwareentwickler entwickelt nicht nur die Software im Sinne eines empirischen Ansatzes, sondern sollte auch einen solches Mindset haben. Er will das Produkt stets verbessern, dies macht er in kleinen Schritten und reflektiert diese zwischendurch. Genau diese Einstellung sollte er auch zu seinem Handwerk haben. Er will sich stets verbessern, stets neue Sachen und Herangehensweisen lernen. Er kann sein erlerntes Wissen weitergeben aber auch neue Sachen von anderen annehmen.

### Entwicklermentalitäten

Erfahrene Entwickler äussern ihre Erfahrung auf verschiedene Arten. Das eine extrem ist, die “sich tendenziell überschätzende” Art. Für diese Entwickler ist alles relativ einfach umzusetzen. Probleme scheint es nicht zu geben, nur Lösungen, und kaum ist ein Problem andiskutiert wissen diese Entwickler bereits wie man sie am besten angehen und lösen sollte. Das andere extrem ist die “sich tendentiell unterschätzende” Art. Diese Programmierer sehen überall Herausforderungen. Werden Probleme andiskutiert, so sind für diese Entwickler erstmal nur eine Menge an Problemen zu sehen. Generell denke ich, dass die zweite Art eher für ein agiles Entwicklungsteam geeignet ist. Sie nehmen Probleme ernst, was dazu führt dass sie sich damit auseinandersetzen. Sie hinterfragen die Dinge eher als die Entwickler, welche schon die vermeindliche Lösung im Kopf haben.

Entwickler die sich in solchen Situationen tendenziell überschätzen neigen dazu, Aufwandschätzungen etc. zu beinflussen. Diskussionen mit Product Owner oder Stakeholder sind zudem nicht so konstruktiv, da viele Lücken in den Anforderungen erst durch nachfragen der Entwickler auftauchen. Das Nachfragen und hinterfragen macht man jedoch nur, wenn man ein Problem sieht, nicht wenn man schon die geeignete Lösung dafür im Kopf hat. **Ein optimales Team sollte meiner Meinung nach jedoch von beiden Mentalitäten Vertreter haben**.

## Schlussfolgerung

Ein Agile Entwickler denkt nicht in Projekten oder Meilensteinen. Ein Agiler Entwickler hat stets das Produkt im Fokus und will gute Lösungen für den Anwender entwickeln. Er ist Teamfähig, kann mit verschiedensten Rollen kommunizieren, ist technisch sehr versiert und arbeitet sich in den Fachbereich der Software hinein. All dies macht den Agilen Softwareentwickler zu einer sehr wertvollen Person im Unternehmen.

Für Unternehmen stellt sich die Frage, wie kommt man an solche Agile Softwareentwickler. Volker Birk erläutert dies in seinem Talk[[10]](#footnote-39) ganz gut: **“Du musst sie ausbilden, ab und zu bekommst du einen geschenkt”**.

## Referenzen

1. Agiles Manifest, agilemanifesto.org [↑](#footnote-ref-22)
2. Clean Coder, Robert C. Martin, Amazon [↑](#footnote-ref-25)
3. Extreme Programming - Eine Übersicht, Rolf Dornberger & Thomas Habegge, Fachhochschule Solothurn Nordwestschweit, Reihe A: Paper 2005-05 [↑](#footnote-ref-26)
4. Der leidenschaftliche Programmierer, Chad Fowler, Amazon [↑](#footnote-ref-28)
5. Domain-Driven Design, Eric Evans, Amazon [↑](#footnote-ref-29)
6. Extreme Programming Explained, Kent Beck, Amazon [↑](#footnote-ref-30)
7. Software Engineering, Volker Birk, YoutTube [↑](#footnote-ref-32)
8. Agiles Manifest, agilemanifesto.org [↑](#footnote-ref-33)
9. Domain-Driven Design, Eric Evans, Amazon [↑](#footnote-ref-34)
10. Software Engineering, Volker Birk, YoutTube [↑](#footnote-ref-39)