WIE GELINGT LEAN SOFTWARE DEVELOP-MENT?

Ein Artikel von Josef Wohlrab, ROI-EFESO

Bereits Ende der 1980er Jahre trat die Lean-Philosophie ihren Siegeszug durch die Industrie an. Sie entstammt der Produktionsorganisation von Toyota. Ihr Kern – japanisch „Muda" – ist das stetige Streben nach Vermeidung jeder Art von Verschwendung, was sowohl im Sinne von Effektivität als auch von Effizienz zu verstehen ist.

Auch wenn die Lean-Prinzipien ihren Ursprung in der Industrie haben, wurde schnell deutlich, dass der Erfolg von Toyota nicht allein auf ihrer Anwendung in der Produktion beruht. Vielmehr prägte das schlanke Denken und Arbeiten auch alle anderen Unternehmensbereiche, so auch die Produktentwicklung. Bereits 2003 befassten sich die IT-Forscher und Programmierer Mary und Tom Poppendieck mit der Anwendung der Lean-Philosophie auf die Softwareentwicklung. Dabei hatten sie vor allem die „reine" Softwareentwicklung im Blick. In den vergangenen Jahren ist diese Perspektive weiterentwickelt worden, sodass sich die Idee schlanker Produktentwicklung auch im Kontext komplexer Systeme und aktueller Technologien wie IoT (Internet of Things), Cloud oder Augmented Reality anwenden lässt. Dabei haben sich fünf wesentliche Prinzipien herauskristallisiert, die im Folgenden näher betrachtet werden:

1. Kundennutzen definieren und maximieren
2. Wertstrom identifizieren und Abfall beseitigen
3. Wertstrom fließen lassen
4. Team stärken
5. Kontinuierlich lernen und verbessern

Kundennutzen definieren und maximieren

Für jedes Produkt ist der wahrgenommene Kundennutzen im Vergleich zu den Wettbewerbsprodukten das entscheidende Erfolgskriterium. Eine Entwicklung am Nutzer vorbei ist daher die schlimmste Form der Verschwendung. Um dies zu vermeiden, gilt es, den tatsächlich wahrgenommenen Wert frühestmöglich zu validieren und weitere Kundenbedürfnisse zu ermitteln. Dies gelingt über die schnelle Erzeugung eines Minimum Viable Product (MVP) und das Einholen von Kundenfeedback. Die Produktveröffentlichung aufzuschieben, um eventuell mehr Features bringen zu können, erhöht deutlich das Risiko, den durch den Kunden wahrgenommenen Nutzen oder das Window of Opportunity im Markt zu verfehlen, weil Wettbewerber alternative Angebote in den Markt bringen. Diese müssen noch nicht einmal einen vergleichbaren Funktionsumfang oder die gleiche Qualität aufweisen, wenn sie entweder auf einen dringenden Bedarf treffen oder die Wechselwilligkeit der Kunden gering ist. Geschwindigkeit in Form von schnellem Prototyping und engzyklischer Kundeneinbindung ist also entscheidend. Dadurch wird auch das Risiko von Over-Engineering deutlich reduziert.

Um den Kundennutzen schnell zu maximieren und keine Zeit zu verlieren, kommen Methoden wie das A/B Testing zum Einsatz, bei dem gleichzeitig mehrere Varianten entwickelt und getestet werden, sodass durch das Verhalten der Nutzergruppen die bessere Variante erkennbar wird. Diese Methoden erscheinen aus der Perspektive klassischer Produktion wie reine Verschwendung, sind aber im Kontext der Softwareentwicklung genau das Gegenteil.

Wertstrom identifizieren und Abfall beseitigen

Ziel einer Wertstromanalyse und -optimierung ist es, den kritischen Pfad zu identifizieren und bestmöglich zu komprimieren sowie nicht wertschöpfende Aktivitäten zu eliminieren. Der kritische Pfad beschreibt die Mindestdauer der Feature-Entwicklung. Er ergibt sich aus dem Entwicklungsumfang und dem angewandten Entwicklungsmodell bei idealen Konditionen. Bei der Entwicklung von Systemen schließt dies auch die Entwicklung der benötigten Hardware mit ein. Häufig wird die Software nach einem agilen Modell entwickelt und die Hardware nach einem V-Modell. Um eine volle Ausrichtung auf den Wertstrom und den enthaltenen kritischen Pfad zu realisieren, müssen die beiden Welten in einem hybriden Modell verzahnt werden. Da das passende Entwicklungsmodell immer vom unternehmensspezifischen Kontext abhängt, sollten bekannte Modelle wie SAFe (Scaled Agile Framework) oder LeSS (Large Scale Scrum) nicht einfach kopiert, sondern immer an die eigenen individuellen Anforderungen angepasst werden.

Doch die Gestaltung des besten Modells ist wertlos, wenn es nicht gelebt wird. Ein fehlende Durchgängigkeit in der Anwendung hat oft Blindleistungen, Wartezeiten oder Qualitätsprobleme zur Folge. Die Implementierung erfordert deshalb einen klaren Fokus. Die Visualisierung des Wertstroms und konkrete Praxisbeispiele haben sich sowohl für die Optimierung des Wertstroms als auch für dessen Vermittlung bewährt. Um die geplante Entwicklung der Features abzusichern, werden dabei technische Reviews entlang des kritischen Pfads installiert.

Nicht selten werden diese Reviews jedoch überdimensioniert, da sie aufgrund der technischen Komplexität nicht mehr in der Verantwortung und Kompetenz einzelner Personen liegen können. Schlechte oder fehlende Architekturarbeit und ein schlechtes Schnittstellenmanagement (API-Management) können diese Situation weiter verschlimmern. Ein weiteres Phänomen ist der leichtfertige Umgang mit der Softwarereife zu den vereinbarten Meilensteinen, da sich Software ohne augenscheinlich große Kosten anpassen lässt. Dadurch häufen sich technische Schulden an und werden zunehmend zu einem schwer lösbaren Problem. Gut aufgesetzte und gelebte Qualitätspraktiken sowie ein stringenter Umgang mit Abweichungen von der vereinbarten Softwarereife sollten deshalb unbedingt gewährleistet werden.

Wertstrom fließen lassen

Der Wertstrom und das zugrunde liegende Prozessmodell betten sich in die Struktur einer Entwicklungsorganisation ein, mit entsprechenden Berichts- und Entscheidungswegen. Diese Elemente sind idealerweise stimmig zueinander und voll auf den Wertstrom ausgerichtet. Doch Entwicklungsorganisationen haben selten nur einen Wertstrom zu bedienen und sind üblicherweise nicht beliebig schnell und oft veränderbar. Daher ist es logisch, dass beim Streben nach einem globalen Optimum die Rahmenbedingungen für den einzelnen Wertstrom nicht ideal sein können. Der Fachkräftemangel tut dabei ein Übriges, wenn versucht wird, mit einer stark unterbesetzten Mannschaft das volle Produktportfolio zu bedienen. Die unzureichende Priorisierung führt zu einer Überbuchung bzw. Mehrfachzuordnung von Mitarbeitern zu parallel laufenden Projekten und zu einer zu hohen Zahl an Projektübergaben. Dadurch wird die Leistungsfähigkeit der Entwicklungsorganisation drastisch reduziert. Das Management reagiert darauf oftmals mit mehr störenden als helfenden Interventionen, was neben der fehlenden Fähigkeit zu priorisieren ein deutlicher Hinweis für Defizite in der Führungskompetenz ist. Darüber hinaus zwingt ein derart stark überlastetes System die Führungskräfte in einen reaktiven Modus. Präventives, gestaltendes Handeln und ein gutes Risikomanagement bleiben dabei typischerweise auf der Strecke.

Team stärken

Softwareentwicklung erfolgt üblicherweise in agilen Teams. Im Idealfall ermöglicht die Zusammensetzung der Teams ein weitgehend autonomes Bearbeiten der Aufgabenumfänge. Das schließt die Ermächtigung zur Entscheidung, das Empowerment, mit ein, was voraussetzt, dass alle benötigten fachlichen und sozialen Kompetenzen im Team vertreten sind. Die übertragene Entscheidungskompetenz fördert die Eigenverantwortung und steigert die Motivation. Somit können agile Teams in einem dynamischen Umfeld, d.h. bei sich stetig verändernden Anforderungen, besonders produktiv agieren. Die Führungskraft spielt hierbei eine komplett andere Rolle als im klassischen Modell. Ihre wesentliche Aufgabe ist es, optimale Rahmenbedingungen zu schaffen sowie gleichzeitig die Governance zu stärken und die Accountability sicherzustellen.

Darüber hinaus gibt es in einem agilen Set-up Rollen, die für die Performance der Teams von besonderer Bedeutung sind: Architects, Product Owners und Agile Masters bzw. Scrum Masters. Der Architekt ist dabei als inhaltlich tragende Rolle hervorzuheben, da die Architektur nicht nur für die Lebensdauer von Software entscheidend ist, sondern auch für die verteilte Entwicklung, d.h. die Parallelisierung der Entwicklungsaktivitäten. Um diese Rolle einnehmen zu können, braucht es sowohl eine hohe fachliche als auch Führungskompetenz. Der Product Owner richtet die Softwareentwicklung maximal auf den Kundennutzen aus. Der Agile Master hat die Aufgabe, die Zusammenarbeit zu optimieren. Er verantwortet die Einhaltung und Verbesserung des agilen Entwicklungsmodells sowie die Beseitigung von Hindernissen („Impediments“ in der agilen Nomenklatur) jeglicher Art.

Kontinuierlich lernen und verbessern

Kontinuierliches Lernen und Verbessern ist ein integraler Bestandteil agiler Frameworks. Das Streben nach kontinuierlicher Verbesserung kann jedoch nicht erzwungen werden, wie zahlreiche Praxiserfahrungen zeigen. Eine gute Chance auf nachhaltigen Erfolg besteht erst dann, wenn die kontinuierliche Verbesserung zu einem integralen Bestand der Unternehmenskultur wird. Dafür sind insbesondere eine offene Fehlerkultur und ein kollaboratives Mindset starke Indikatoren. In einer offenen Fehlerkultur setzen sich alle Beteiligten aktiv und konstruktiv mit Fehlern auseinander, um von ihnen zu lernen und sich weiterzuentwickeln. Die Offenheit setzt hierarchieübergreifend ein großes Vertrauen voraus. Bei der Zusammenarbeit steht das übergeordnete Ziel und nicht die individuellen Ziele im Vordergrund. Man unterstützt sich gegenseitig und teilt proaktiv Wissen, um jenseits hierarchischer Selbstverständnisse und Silo-Strukturen schneller und besser zum gemeinsamen Ziel zu kommen. Gerade in der Softwareentwicklung stellt die Pflege einer solchen Fehler- und Zusammenarbeitskultur oftmals eine Herausforderung dar, die sich im multikulturellen Umfeld zusätzlich verschärfen kann. Teambuilding und vertrauensbildende Maßnahmen sind daher sehr wichtig, kommen jedoch bei hohem Lieferdruck mit einer unterbesetzten Mannschaft oftmals zu kurz. Neben den „soften“ Themen, die den wesentlichen Stellhebel darstellen, lassen sich auch Messgrößen definieren, anhand derer gesteuert werden kann. Diese sind aber immer nur so gut wie die zugrunde liegende Datenbasis und wirken bei schlechter Wahl kontraproduktiv.

Vor dem Hintergrund dieser Betrachtung wird deutlich, dass es viele Stellhebel für Lean Software Development gibt. Sie alle dienen dazu, Verschwendung und Verbesserungspotenziale zu erkennen. Externes Feedback erweist sich dabei als besonders hilfreich, um einseitigen Perspektiven und Betriebsblindheit entgegenzuwirken und den Blick für das Ganze nicht zu verlieren.