IT-Sachverständiger mit Praxiserfahrungen gibt Empfehlungen zum Rollenmodell bei agiler Softwareentwicklung

Wie kann mein Kunde bzw. mein Auftraggeber „König" sein und bleiben?

Es folgen Vorschläge zum Zuordnen von Verantwortungen analog dem Rollenmodell bei agiler Projektarbeit für ein Softwarehaus.

(In diesem Text wird aus Gründen der Lesbarkeit und Klarheit auf die Verwendung gendergerechter Sprache verzichtet.)

I. Product Owner versus Kundenprojektleiter

Erste These:
Das agile Vorgehensmodell muss die Rolle und die Einschränkung von Kompetenzen für den **Kundenprojektleiter** meines Unternehmens viel mehr verankern, als dies gemeinhin in der Praxis erfolgt.

Risiko:
Kundenwünsche werden individuell umgesetzt und zerstören die saubere Struktur meiner Softwarebasis.

Strategie:
Mein Softwarehaus muss wirtschaftlich agieren. D.h.: Die Lösung, die ein Kunde von mir erhält, sollte mittels der Nutzung meiner Software-Basis implementiert werden. Die Erweiterung dieser Basis wird sukzessive erfolgen, sollte aber immer mit Blick auf „meinen" Softwarestandard* erfolgen.

*Softwarestandard: Hier ist von der eingerichteten Entwicklungsumgebung über Architektur, Bibliotheken bis hin zur Oberfläche die Ergebnisse der bisherigen Entwicklungsarbeiten gemeint. Diese Ergebnisse sind neben dem Humankapital das wichtigste Asset meines Unternehmens!

Mein Kundenverantwortlicher bzw. Kundenprojektleiter – diese Rolle ist im agilen Vorgehensmodell nach Ansicht des Verfassers nur unzureichend abgebildet – muss seine Vorschläge zur Abbildung von Kundenwünschen in der Software eng mit dem Entwicklungsleiter bzw. dem **Product Owner** abstimmen.

Andernfalls „zerfleddert" meine Basis, jeder Kunden bekommt eine „Extrawurst" und mein Second-Level-Support kann – wenn Befunde vom Kunden gemeldet werden - nicht mehr effizient arbeiten.

Beachten:
Als Ziel muss mit dem Kunden vereinbart werden, dass Umsetzungen seiner Sonderwünsche so spezifiziert werden, dass ein neues Modul an alle Kunden ausgeliefert werden kann. Ggf. wird dann über das Berechtigungskonzept das neue Modul nur bei dem Anforderer freigeschaltet. Die anderen Kundenverantwortlichen erhalten die Funktionsbeschreibung des neuen Moduls als Vertriebsinformation und analysieren, ob sie es für den eigenen Kunden nutzbringend verkaufen können.

II. Product Owner versus Entwicklungsteam*:

*__Entwicklungsteam à la Agil:__ Ein selbstorganisiertes und interdisziplinäres Team, das die Arbeit aus dem Product Backlog umsetzt. Das Team ist für die Lieferung von Inkrementen funktionsfähiger Software verantwortlich.

Zweite These:
Die Rolle „Entwicklungsleiter" (im Fokus hier „**Technical Lead**", d.h. Chef der Architektur und der Funktionsbibliotheken) ist im agilen Vorgehensmodell nicht exakt genug definiert.

Risiko:
Dem Team werden zu viele Freiheiten eingeräumt.

Strategie:
Wer Softwareentwicklung geleitet hat kennt das folgende Szenario: ein findiges Teammitglied ist mit seiner Aufgabe überraschend schnell fertig geworden. Als **Technical Lead** schauen Sie sich das Ergebnis genauer an und stellen fest, dass hier neue Bibliotheken genutzt wurden, die Ihnen noch nicht bekannt waren. Meist ist das sogenannte Open Source Software, die man in der Entwicklung einbinden kann. Allerdings hat Open Source ganz verschiedene Lizenz-Bedingungen. Eine kostenlose Nutzung kann mit einem Update passé werden und kann damit die Kostenkalkulation unvorhersehbar beeinflussen. Die Erweiterung mit neuen Bibliotheken bringt auch eine Erweiterung von neuen, oft noch unbekannten Schwachstellen mit sich.

Beachten:
Bei der Unmenge von Drittsoftware muss die Aufgabe „Bibliotheksverwaltung" unbedingt in eine verantwortliche Hand gegeben werden. Die Teams haben sich nur an freigegebenen Bausteinen zu bedienen. Neue Bausteine müssen beim **Technical Lead** beantragt werden. Ob diese Arbeit der **Product Owner** oder ein benannter **Stakeholder** leistet? Egal, aber diese Aufgabe muss unbedingt adressiert sein.

III. Bedienphilosophie und Usability versus Agile Anpassungen

Dritte These:
Eine Bedienphilosophie muss den heutigen Standards genügen und sich über die gesamte Software „ziehen".

Risiko:
Wer kennt es nicht? Jede „Ecke" in der Software verlangt eine andere Bedienung. An der einen Stelle kann ich mit einem Rechtsklick auf ein Objekt agieren, an der nächsten Stelle muss ich irgendwelche Buttons finden und in einem dritten Bereich komme ich ohne First-Level-Support nicht weiter.

Wir reden hier über Usability: keiner hat heute die Zeit und bzw. die Akzeptanz, sich in exotische Bedienphilosophen einzuarbeiten.

Strategie:

Die für die User-Interaktion verwendeten Bibliotheken sollten auf Usability geprüft sein und den verbreiteten Standards entsprechen.

Beachten:
Definieren Sie die Bedienphilosophie! Es existieren heute Pseudo-Industrie-Standards, die von den bekannten Systemen wie Amazon, PayPal usw. gesetzt werden. Als Unternehmen in der Softwareindustrie muss ich organisieren, dass jeder in meinen Teams die (im Zielland) bekannten und genutzten Standards kennt.
Organisieren Sie eine „Kundenreise", d.h. lassen Sie Ihre Software von unbedarften Personen bedienen. Zwingen Sie Ihre Entwickler, sich mit den gängigen Pseudo-Standards auseinanderzusetzen um damit kreative, aber exotische Eigenentwicklungen zu vermeiden.

IV. Ausnahmebehandlungen und Stabilität bzw. Handlungssicherheit

Vierte These:
Kein Entwickler ist in der Lage, jedes mögliche Fehlerszenario bei der Verarbeitung von Eingaben, Korrekturen, copy- and paste-Aktionen und Rücksprüngen in den verschiedenen Browsern vorauszusagen.

Risiko:
Auch diesen Fall hat wahrscheinlich schon jeder einmal erlebt: Die Software hat sich „verklemmt" und nur nach einem harten Restart (im Fall des Falles Gerät aus- und wieder einschalten) kann ich meine Arbeit mit dem System fortsetzen.

Strategie:
Es müssen Funktionsbausteine bereitgestellt werden, die eine Ausnahmebehandlung ermöglichen. Exception Handling bedeutet, dass bei einem Fehler nach Verarbeitung von Daten das System in einem Zustand zurückgebracht wird, der vor der Verarbeitung (dem Fehler) bestand.

Beachten:
Hier müssen mehrere Techniken (am besten per Code implementierte Lösungsbeispiele) bereitgestellt werden, die bei einer Datenverarbeitung sowohl eine Vorbereitung vornehmen (Prüfung der Möglichkeit der Verarbeitung), die Verarbeitung selbst kontrollieren und in einer Nachbereitung einen Check auf das Ergebnis vornehmen.
Damit wird eine einfache Funktion natürlich relevant „aufgebläht", aber ein komplexeres Softwaresystem kann nur so stabilisiert und handlungssicher werden.

Jeder Entwickler muss mit der Anwendung dieser Funktionen vertraut sein!

Fazit

Wenn ich als „Softwarehaus" (oder als Inhouse-Abteilung) Erfolg haben will, muss ich meine Kunden jederzeit und bedürfnisgerecht bedienen können.

Das funktioniert nicht für ein individuelles Softwareprodukt, welches agil von sich selbst steuernden Teams an „irgendwelche" Kundenwünsche angepasst und nur von 1 bis 2 Entwicklern meiner Company gerade so verstanden wird.

Das funktioniert nur, wenn ich meine Softwarebausteine strengstens auf Anpassbarkeit getrimmt habe und alle Kundenwünsche in dieser Anpassbarkeit abbilde.

Die Abbildung der realen Objekte und Prozesse – also der digitale Zwilling – muss in meine Funktionsbibliothek passen. Ist dies nicht der Fall, muss ich ermitteln, wie ich eine Erweiterung konzipiere, so dass diese allen meinen Kunden zugutekommen kann.

Eine stringente Führung der Teams aus Sicht der Nutzung einer möglichst einheitlichen Art und Weise bei der Umsetzung der Aufgaben (Nutzung der Bedienphilosophie, der Bibliotheken und weiterer interner Programmiervorgaben) ist erforderlich.