LEXIKON DER IT-MISSVERSTÄNDNISSE: AGILE SOFTWARE-ENTWICKLUNG VON A BIS Z

der autor

Thomas Epping
(thomas.epping@co-in.de)
arbeitet als Berater für methodische Projektorganisation bei der Cologne Intelligence GmbH. Er interessiert sich für begründete Konzepte als Grundlage für eine solide Softwareentwicklung.

„Wir brauchen das *Commitment* von allen, unser Projekt ab jetzt *agil* zu gestalten. Wir müssen *Ballast* abwerfen, ohne *Details* zu verlieren. Dann kann *eigentlich* nichts mehr schief gehen." Klingt energisch, aber was ist gemeint? Dieses fiktive Zitat enthält die ersten fünf Wörter aus einem Lexikon der Missverständnisse, von dem dieser Artikel handelt. Er möchte auf Fallstricke beim Gebrauch gängiger, meist agiler Vokabeln aufmerksam machen und dabei helfen, den Projektalltag klarer zu gestalten.

Die Softwareentwicklung ist keine exakte Wissenschaft. Viele ihrer Vokabeln bieten Raum für Interpretationen und Missverständnisse. Natürlich sind diese niemals ganz zu vermeiden, aber es ist schon ärgerlich, wenn im Projektalltag Erwartungen voneinander abweichen, obwohl in Gesprächen zuvor scheinbar nur klare Vokabeln benutzt worden sind. Ich lade Sie zu einem Streifzug durch ein Lexikon ein, das in Einträgen von A bis Z auf die Gefahr von Missverständnissen bei gängigen Vokabeln hinweisen soll. Die meisten von ihnen sind agil geprägt, denn eine klare gemeinsame Sprache fördert Agilität. Jeder Eintrag enthält eine kurze Anregung, wie Sie dem beschriebenen Missverständnis in Ihrem Projekt vorbeugen können.

Von Agil bis Fehler

Agil ist ein Wort, das im Laufe der letzten zehn Jahre in vermutlich so gut wie jedem Projekt einmal gefallen ist. Seine Grundlage ist das Agile Manifest (vgl. [Agi]), dessen Verfasser abwägen: „[...] we have come to → value [...] over [...]." Die Formulierung als subjektive Abwägung macht es jedoch unmöglich, das agile Manifest als objektive Definition für Agilität zu verwenden. Wann ist ein Projekt agil? Ein Vorschlag: Nennen Sie Ihr Projekt agil, wenn Sie sich – vor die Wahl gestellt – immer für den Weg entscheiden, dem auch die Autoren des Agilen Manifests mehr Wert als seiner Alternative zugesprochen haben:

- Individuen und Interaktionen mehr als Prozesse und Werkzeuge.
- Funktionierende Software mehr als umfassende Dokumentation.
- Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung.

Ein Wort kann ein Missverständnis auslösen, wenn es nicht genau definiert oder seine Definition kontextabhängig ist.
Beispiele aus diesem Lexikon: Agil, Ballast, Commitment, Detail, Jetzt, Manager, Priorität, Value.

Kasten 1: Missverständnisse durch Ungenauigkeit.

- Reagieren auf Veränderung mehr als das Befolgen eines Plans.

Dieses Wort aus dem Lexikon ist ein Beispiel für ein Missverständnis durch Ungenauigkeit, weitere Beispiele sind in Kasten 1 aufgeführt.

Ballast (*Waste*) bezeichnet alles, was nicht von Wert (→ *Value*) für einen Kunden ist. Erst in Abhängigkeit davon, was als wertvoll betrachtet wird, kann also entschieden werden, was Ballast ist. Zum Beispiel muss es bei einer Anforderung mit hohem Risiko kein Ballast sein, mehrere Lösungsmöglichkeiten gleichzeitig zu entwickeln, wenn die Risikominimierung als wertvoll für den Kunden betrachtet wird – dieses Vorgehen ist als „Set-Based Design" bekannt (vgl. [Pop07]). Bei einer Anforderung mit geringem Risiko kann die Entwicklung von mehreren Lösungsmöglichkeiten jedoch Ballast sein. Seien Sie sich sowohl des Ballasts als auch des Werts Ihrer Entscheidungen für Ihren Kunden bewusst (siehe Kasten 2).

Mary und Tom Poppendieck haben sieben Arten von Ballast in der Produktion auf die Softwareentwicklung übertragen (vgl. [Pop07]).
Ballast in der Produktion: In-Process Inventory; Over-Production; Extra Processing; Transportation; Motion; Waiting; Defects.
Ballast in der Softwareentwicklung: Partially Done Work; Extra Features; Relearning; Handoffs; Task Switching; Delays; Defects.

Kasten 2: Sieben Arten von Ballast.

Commitment ist durch das → agile Vorgehensmodell Scrum bekannt geworden und hat seitdem viele Missverständnisse ausgelöst. Ist damit eine lose Absicht, ein Versprechen oder eine zwingende Verpflichtung gemeint, ein → Ziel zu erreichen? Inzwischen haben die Väter von Scrum, Ken Schwaber und Jeff Sutherland, diesen Missverständnissen Rechnung getragen. Im aktuellen Scrum Guide sprechen sie nicht mehr von „Commitment", sondern von „Forecast" (vgl. [Scru]). Machen Sie in Ihrem Projekt die Erwartungshaltung für Ziele klar.

Detail ändert, ähnlich wie → Value, seine Bedeutung mit dem Kontext. Der nötige Detailgrad für eine Anforderung an eine Anwendung kann sich zum Beispiel in Abhängigkeit von ihrer Umsetzungsphase ändern. Muss eine Anforderung detailliert genug sein, um sie planen, schätzen oder entwickeln zu können? Über den Detailgrad, der für jeden Kontext (mindestens und höchstens) erforderlich ist, sollte Einigkeit bestehen. Ein Beispiel für eine kontextabhängige Detaillierung ist das Backlog in Scrum, in dem nach oben sortierte Einträge zur Schätzung bereit und daher detaillierter sind als nach unten sortierte Einträge, die vorerst nur zur groben

Mary und Tom Poppendieck nennen **sieben Grundsätze der schlanken Softwareentwicklung** (vgl. [Pop07]).

- Eliminate Waste
- Build Quality In
- Create Knowledge
- Defer Commitment
- Deliver Fast
- Respect People
- Optimize the Whole.

In diesem Lexikon werden einige davon aufgegriffen: Ballast, Qualität, Information, Commitment, Jetzt und Optimierung.

Kasten 3: *Grundsätze der schlanken Softwareentwicklung.*

Planung genutzt werden (vgl. [Pic08]). Fragen Sie bei der Forderung, „Das brauche ich detaillierter“, einmal nach: „Um was damit machen zu können?“

Eigentlich ist, zugegeben, kein missverständliches Wort, kann aber ein Indikator für Missverständnisse sein. Es macht Unsicherheit deutlich: „Eigentlich müsste die Anforderung → jetzt umgesetzt sein.” Ein Grund für diese Unsicherheit kann die Vermutung sein, dass es einen Unterschied zwischen dem eigenen Verständnis der Anforderung und dem von anderen gibt. In diesen Fällen kann das Wort als Auslöser genutzt werden, um das Verständnis der Anforderung zwischen allen beteiligten Personen noch einmal abzugleichen. Wenn „eigentlich“ ein häufiges Wort in Ihrem Projekt ist, prüfen Sie, ob Sie Ihre Projektorganisation so ändern können, dass diese Unsicherheit verschwindet.

Fehler sind nichts, was eine Anwendung aufweist oder eine Person gemacht hat, sondern etwas, was eine Organisationsform zugelassen hat. Um das zu erkennen, kann es nötig sein, den Blick über die konkrete Projektarbeit hinaus zu erweitern: Sie sind mit den Leistungen der Personen in ihrem Projekt nicht zufrieden? Vielleicht müssen Sie überdenken, wie Sie derzeit die Weiterbildung oder die Anstellung von Personen organisieren. Gestalten Sie Ihr Projekt fehlertolerant und versuchen Sie, aus Fehlern → Information zu gewinnen. Benutzen Sie Fehler nicht als Druckmittel. Dieses Wort aus dem Lexikon ist ein Beispiel für ein Missverständnis durch

Das Cynefin-Framework unterscheidet zwischen den folgenden Typen von Aufgaben (allgemeiner: Systemen) und zugehörigen Herangehensweisen (vgl. [Cyn]).

- **Simple:** Sense – Categorize – Respond. In einfachen Systemen ist der Zusammenhang zwischen Ursache und Wirkung offensichtlich. Ihnen wird mit *Best Practices* begegnet.
- **Complicated:** Sense – Analyze – Respond. In komplizierten Systemen ist der Zusammenhang zwischen Ursache und Wirkung nicht offensichtlich, aber vorhanden. Ihnen wird mit *Good Practices* begegnet.
- **Complex:** Probe – Sense – Respond. In komplexen Systemen ist der Zusammenhang zwischen Ursache und Wirkung nur nachträglich feststellbar. Ihnen wird mit *Emergent Practices* begegnet.
- **Chaotic:** Act – Sense – Respond. In chaotischen Systemen gibt es keinen Zusammenhang zwischen Ursache und Wirkung. Ihnen wird mit *Novel Practices* begegnet.

Zusätzlich gibt es den Zustand von **Disorder**, in dem sich ein System solange befindet, wie nicht klar ist, welchem der vier Typen es zugeschrieben werden kann.

Kasten 4: *Typen im Cynefin-Framework.*

Manipulation, weitere Beispiele sind in Kasten 7 aufgeführt.

Von Geld bis Leicht

Geld in Form von Prämien oder andere Mittel zur extrinsischen Motivation (das können zum Beispiel → Team-Events oder → Ziel-Vereinbarungsgespräche sein) sind immer noch in Projekten anzutreffen. Mehr und mehr scheint sich allerdings die Erkenntnis durchzusetzen, dass intrinsische Motivation die nachhaltigere Form der Motivation ist. Während extrinsische Motivation die Gefahr der kurzfristigen lokalen → Optimierung in sich trägt, wird der intrinsischen Motivation ein langfristig ganzheitliches Potenzial zugesprochen. Fördern Sie diese, indem Sie jedes Ziel in Ihrem Projekt begründen und es Ihrem Team damit ermöglichen, sich mit ihm zu identifizieren.

Helden untergraben die Zusammenarbeit in einem Projekt. Das Wort ist zwar positiv

Ein Wort kann ein Missverständnis auslösen, wenn es in der Annahme verwendet wird, dass das eigene Verständnis auch das von anderen ist.
Beispiele aus diesem Lexikon: Helden; Optimierung; User; Wasserfall; Qualität.

Kasten 5: *Missverständnisse durch eine nicht validierte Annahme.*

besetzt, aber auch ein Signal für Engpässe. Entsprechend ist jede Heldentat ein kurzfristig umgangener Engpass und eine lediglich lokale → Optimierung. Helden verhindern nachhaltige Lösungen, sodass anstelle von Heldentaten auch von einem fortdauernden Martyrium (für Personen und Projekte) gesprochen werden kann (vgl. [Rot05]). Vermeiden Sie Helden, indem Sie zum Beispiel auf → Geld oder andere Mittel zur extrinsischen Motivation verzichten und Engpässe in Ihrem Projekt systematisch auflösen. Dieses Wort aus dem Lexikon ist ein Beispiel für ein Missverständnis durch eine nicht validierte Annahme, weitere Beispiele sind in Kasten 5 aufgeführt.

Information ist nicht identisch mit Beobachtung, Daten oder Wissen (vgl. [Sim11]). Aus einer Beobachtung gehen zunächst lediglich Daten hervor. Erst, wenn diese beobachteten Daten in einen Kontext gestellt und damit interpretiert werden, ergibt sich Information. Angewandte, in die Praxis getragene Information schließlich liefert Wissen. Reine Kennzahlen aus Ihrem aktuellen Projekt sind also keine Information. Wenn Sie diese Kennzahlen jedoch interpretieren und die daraus resultierende Information anwenden, haben Sie sie in Wissen umgewandelt. Etablieren Sie für diese Transformation zum Beispiel eine → Retrospektive in Ihrem Projekt. Dieses Wort aus dem Lexikon ist ein Beispiel für ein Missverständnis durch Unkenntnis, weitere Beispiele sind Kasten 6 aufgeführt.

Jetzt, wirklich jetzt? Eile mit Weile – stellen Sie sicher, dass Ihre Eile auf die → Priorität Ihrer Anforderungen abgestimmt ist. Wenn sich hinter einer hohen Priorität eine Anforderung mit einem festen Termin verbirgt, kann Eile angebracht sein; wenn sich hinter einer hohen Priorität eine Anforderung mit hohem Risiko verbirgt, kann Weile angebracht sein. Fragen Sie sich, ob „jetzt” wirklich der geeignete Moment für Ihre Handlungen ist. Der Grundsatz „Defer → Commitment“ ist

Ein Wort kann ein Missverständnis auslösen, wenn es ohne Kenntnis einer bestehenden Definition verwendet wird.
Beispiele aus diesem Lexikon: Information, Komplex, Leicht, „Nur was man messen kann, kann man auch managen“, Retrospektive, „X never, ever marks the spot“, Ziel.

Kasten 6: Missverständnisse durch Unkenntnis.

übrigens einer der sieben Grundsätze der schlanken Softwareentwicklung (siehe Kasten 3).

Komplex ist nicht kompliziert. Der Unterschied zwischen diesen beiden Wörtern ist wichtig, denn komplexe Aufgaben erfordern eine andere Herangehensweise als komplizierte Aufgaben. Ein wichtiger Unterschied zwischen komplexen und komplizierten Aufgaben ist, dass ein Zusammenhang zwischen Ursache und Wirkung im komplexen Fall nicht a priori festgestellt werden kann, im komplizierten Fall hingegen schon (siehe Kasten 4). Die Softwareentwicklung ist im Allgemeinen eine komplexe Aufgabe. Überlegen Sie daher, in Ihrem Projekt die nähere Vergangenheit zum Beispiel in einer → Retrospektive zu analysieren, um daraus für die Zukunft zu lernen.

Leicht ist eine Aufgabe in Ihrem Projekt wahrscheinlich weniger oft, als Sie meinen, weil die Softwareentwicklung im Allgemeinen eine → komplexe Aufgabe ist. Wäre ihr Projekt (in Worten des Cynefin-Frameworks, siehe Kasten 4) einfach, wären Sie vermutlich nicht damit beauftragt worden. Wäre es chaotisch, hätten Sie den Auftrag vermutlich nicht angenommen. Nehmen Sie Bedenken von Experten aus Ihrem → Team daher immer ernst.

Von Manager bis Retrospektive

Manager müssen ihre Rolle spätestens überdenken, seit die Softwareentwicklung → agil geworden ist. Ihre Rechte und Pflichten ändern sich im Allgemeinen mit dem Vorgehensmodell, das in einem Projekt eingesetzt wird. Ein → Wasserfall-Manager hat andere Aufgaben als ein Scrum-Manager. Manager können sich zum Beispiel nur dann auch als Coach oder Scrum-Master für ihr → Team verstehen (und können auch nur dann uneingeschränkte Akzeptanz erwarten), wenn sie keine Personalverantwortung für die Personen im Team haben. Synchronisieren Sie in Ihrer Rolle als Manager zum Beispiel den von Ihnen erwarteten, den in Ihrem Vorgehensmodell möglichen und den von Ihrem Team tatsächlich gelebten Grad von → Selbstorganisation.

Ein Wort kann ein Missverständnis auslösen, wenn es in der Absicht einer versteckten Beeinflussung verwendet wird.
Beispiele aus diesem Lexikon: Fehler, Geld, Team, Selbstorganisation, Yes!

Kasten 7: Missverständnisse durch Manipulation.

„Nur was man messen kann, kann man auch managen“, heißt es. Eine Messung liefert allerdings solange keine → Information, solange sie nicht in einen Kontext eingebettet ist. Dieser Kontext kann zum Beispiel durch eine → Retrospektive ermittelt werden. In einer Retrospektive werden teils objektive, teils subjektive Beobachtungen (wie zum Beispiel die Entwicklungsgeschwindigkeit) zunächst gesammelt und dann in einen Kontext gestellt (wie zum Beispiel den der Projektereignisse seit der letzten Retrospektive). Erst dadurch werden Maßnahmen für Verbesserungen und ein Management möglich. Reservieren Sie Zeit in Ihrem Projekt, um den Kontext Ihrer Messungen zu ermitteln.

Optimierung in einer Wertschöpfungskette ist – dies mag kontraintuitiv klingen – nicht in jedem Fall erstrebenswert. Eine lokale Optimierung nur für einen Teil einer Wertschöpfungskette bedingt im Allgemeinen nicht eine globale Optimierung (also eine Verbesserung der gesamten Wertschöpfungskette), sondern kann zu Abhängigkeiten oder Engpässen führen (vgl. [Pop07]). Ein Beispiel dafür ist ein hoch spezialisierter Entwickler, dessen Ausfall (zum Beispiel durch Urlaub, Krankheit oder Kündigung) in einem Projekt vielleicht nur durch → Helden zu verkraften ist. Überlegen Sie vor jeder Optimierung, ob Sie durch sie lediglich einen Teil Ihres Projekts (lokale Optimierung) oder Ihr gesamtes Projekt (globale Optimierung) verbessern.

Priorität kann missverständlich sein, wenn nicht klar ist, aus welchen Kriterien sie sich zusammensetzt. Es kann sein, dass die Priorität einer Anforderung nicht nur durch ein Kriterium (zum Beispiel einen Termin), sondern durch mehrere Kriterien bestimmt wird. So kann es etwa sinnvoll sein, von zwei terminbezogenen Anforderungen die mit dem späteren Termin früher umzusetzen, wenn sie mit einem höheren Risiko verbunden ist. Stellen Sie sicher, dass Ihrem → Team bekannt ist, aus welchen Kriterien sich die Priorität einer Anforderung in Ihrem Projekt zusammensetzt, damit es entscheiden kann, welche Anforderung → jetzt umgesetzt werden muss. Alternativ zur Nutzung einer zusammengesetzten Priorität bietet sich die Einordnung von Anforderungen in Serviceklassen an, wie sie zum Beispiel bei Kanban üblich sind.

Qualität ist gemäß der Norm DIN EN ISO 9000:2005 der „Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt.” Dieser Erfüllungsgrad lässt sich in der Praxis jedoch im Allgemeinen schwer messen: Nicht nur, dass verschiedene Personen in verschiedenen Rollen ihn verschieden interpretieren. Zusätzlich sind neben objektiven Kriterien auch noch subjektive Kriterien von Bedeutung. Aus technischer Sicht mag eine Anforderung erfüllt sein, wenn sie technisch vollständig umgesetzt ist. Aus fachlicher Sicht mag eine Anforderung erfüllt sein, wenn sie zusätzlich alle Anforderungen an die Geschäftslogik erfüllt. Aus Sicht eines → Users können darüber hinaus *Usability* und *User Experience* ausschlaggebend dafür sein, eine Anforderung als mit mehr oder weniger Qualität erfüllt einzustufen. Achten Sie darauf, den Erfüllungsgrad Ihrer Anforderungen für alle Rollen möglichst früh festzustellen und transparent zu machen.

Retrospektive ist, so scheint es, zu einem gern und schnell genutzten Modewort geworden, weil es eine Softwareentwicklung suggeriert, die → agil ist. Eine Retrospektive ist allerdings nicht nur eine einfache Rückschau und auch keine Form von *Lessons learned*, mit der Daten, → Information und Wissen von einem Projekt in das nächste übernommen werden (siehe auch → „Nur, was man messen kann, kann man auch managen”). Vielmehr ist es eine regelmäßige, strukturierte Veranstaltung mit dem → Ziel der Ableitung von Maßnahmen zur kontinuierlichen → Optimierung (vgl. [Der06]). Diese Maßnahmen betreffen das aktuell laufende Projekt. Versuchen Sie, Ihre Maßnahmen klein zu halten und möglichst bis zur nächsten Retrospektive umzusetzen.

Von Selbstorganisation bis Ziel

Selbstorganisation in einem → Team mag ein Wunsch sein, kann allerdings nicht isoliert nur für ein Team geäußert werden. Selbstorganisation geschieht stets in Bezug zu einer Umwelt. Diese beeinflusst Selbstorganisation im Guten wie im Schlechten – im Projekt ist natürlich eine Form gewünscht, die nützlich für das Projekt ist. Versuchen Sie nicht, die Selbstorganisation in Ihrem Team unabhängig von seiner Umwelt zu gestalten. Gestalten Sie stattdessen auch diese, sodass sich die Selbstorganisation in Ihrem Team möglichst frei entwickeln kann. Hören Sie auf die Wünsche, die Ihr Team an seine Umwelt hat.

Team ist ein gern genutztes Wort für die Menge der Personen, die unmittelbar an einem Projekt beteiligt sind. Dabei startet ein Projekt im Allgemeinen mit einer Menge von Individuen, von denen zunächst jedes eher ein eigenes als ein gemeinschaftliches → Ziel verfolgt (Wörter wie „Haufen“ oder „Gruppe“ scheinen manchmal eher angebracht). Fördern Sie den Teamgedanken in Ihrem Projekt durch gemeinsame Ziele und nennen Sie Ihr Team erst dann Team, wenn es auch eines ist. Zählen Sie Ihren Kunden mit dazu?

User, also Anwender, sind nicht unbedingt Customer (also Käufer). Die Unterscheidung zwischen Anwender und Käufer ist wichtig, weil mit ihr zwei verschiedene Entscheidungsgrundlagen für Anforderungen an eine Anwendung verbunden sind. Missverständnisse können entstehen, wenn nicht klar ist, welche dieser beiden Rollen über die → Priorität von Anforderungen entscheidet: Sollen Anforderungen Wert (→ Value) für den Anwender bringen oder Kosten für den Käufer sparen? Anhänger der → agilen Softwareentwicklung haben eine klare Antwort auf diese Frage: Der Wert für den Anwender ist wichtiger. Beobachten Sie, ob Anforderungen in Ihrem Projekt durch Wert für den Anwender oder durch Ersparnis für den Käufer getrieben werden.

Value, also der Wert einer Anforderung für einen → User, ist eng mit der → Priorität einer Anforderung verbunden. Üblicherweise ist die Anforderung mit dem höchsten Wert auch die mit der höchsten Priorität. Entsprechend ihrer Missverständlichkeit kann jedoch auch der Wert einer Anforderung missverständlich sein, nämlich wenn nicht allen Personen im Projekt klar ist, worin dieser Wert → eigentlich besteht. Hier können Personen in verschiedenen Rollen verschiedene Sichtweisen haben und den Wert von Anforderungen (zum Beispiel in Hinblick auf den technischen oder fachlichen Fortschritt) unterschiedlich beurteilen. Überlegen Sie, ob Sie Ihre Anforderungen in Serviceklassen einordnen und damit ihren Wert für alle Personen in Ihrem Projekt transparent machen können.

Wasserfall ist ein Wort, das in dem Artikel, in dem es seinen Ursprung hat, gar nicht erwähnt wird (vgl. [Roy70]). Inzwischen scheint es sich als Synonym für eine schwerfällige Softwareentwicklung durch Command & Control etabliert zu haben. Dabei steht in dem Artikel nicht die Kontrolle von Personen, sondern die Kontrolle von Risiken aufgrund von nachträglichen Anforderungsänderungen im Fokus. Er beschreibt ein Vorgehen, das sehr dokumentenlastig (also nicht sehr → agil) ist, enthält aber auch Forderungen wie die, dass jede Person im Projekt ein elementares Verständnis des gesamten Systems haben und der Kunde fortlaufend in das Projekt eingebunden sein muss. Bedenken Sie bei Ihrem Urteil über den Wasserfall, dass er aufgrund von Erfahrungen mit Anwendungen für Raumfahrtprogramme entstanden ist, also in einer Domäne, die eher kompliziert als → komplex ist. Richten Sie sich bei der Wahl des Vorgehensmodells für Ihr Projekt nicht nach Wörtern, die aus der Mode oder in Mode sind, sondern nach dem, was Sie wirklich brauchen. Stellen Sie Ihr Vorgehensmodell aus Elementen zusammen, die Ihnen und Ihrem → Team angemessen erscheinen, und stören Sie sich nicht daran, wenn diese individuelle Zusammenstellung keinen Namen hat und nicht in eine Schublade passt.

„**X** never, ever marks the spot”, heißt es im Film „Indiana Jones and the Last Crusade”. Auch in der Softwareentwicklung scheint es für kaum ein Projekt eine Schatzkarte mit dem Weg zum → Ziel der perfekten Anwendung zu geben. Dieses Ziel ist beweglich und gleicht oft eher einer schwimmenden Boje auf dem Wasser als einer vergrabenen Truhe an Land. Das ist allerdings kein Grund für die Annahme, dass die perfekte Anwendung nicht zu finden ist. Die Anforderungen an eine Anwendung mögen sich häufig und mit einer nur kurzen Vorlaufzeit ändern, aber moderne Vorgehensmodelle machen eine ebenso häufige und schnelle Reaktion darauf möglich. Prüfen Sie vor dem Projektbeginn und im laufenden Projekt, ob inkrementelle Vorgehensmodelle wie Scrum oder Kanban für Ihr Projekt angebracht sein könnten.

Yes! soll hier für ein bestimmt geäußertes „Ja!“ stehen, das keinen Zweifel aufkommen lässt. Dies kann als Orientierung und Motivation gemeint sein, denn es gibt Bestätigung und einen vermeintlich sicheren Weg in die Zukunft vor: „Brauchen wir diese Anforderung? Yes! Sind wir ein tolles → Team? Yes!“ Doch die Softwareentwicklung ist → komplex, einen sicheren Weg in die Zukunft gibt es im Allgemeinen nicht. Hinterfragen Sie stattdessen die Vergangenheit und prüfen Sie, ob Ihr Team auch dann mit einem bestimmten „Ja!“ antwortet: „Haben wir diese Anfor-▸

Literatur & Links

[Agi] Agiles Manifest, siehe: agilemanifesto.org

[Cyn] Cynefin-Framework, siehe: cognitive-edge.com/library/more/video/introduction-to-the-cynefin-framework/

[Der06] E. Derby, D. Larsen, Agile Retrospektives, Pragmatic Bookshelf 2006

[Pic08] R. Pichler, Scrum, dpunkt.verlag 2008

[Pop07] M. Poppendieck, T. Poppendieck, Implementing Lean Software Development, Addison-Wesley 2007

[Rot05] J. Rothman, E. Derby, Behind Closed Doors, Pragmatic Bookshelf 2005

[Roy70] W.W. Royce, Managing the Development of Large Software Systems, in: Proc. of IEEE WESCON 26, 1970

[Scru] Scrum Guide, siehe: scrum.org/storage/scrumguides/Scrum_Guide.pdf

[Sim11] F.B. Simon, Einführung in die systemische Organisationstheorie, Carl-Auer 2011

derung gebraucht? Waren wir ein tolles Team?“ Werden Sie misstrauisch, wenn Antworten das Wort → eigentlich enthalten, und etablieren Sie eine regelmäßige Verbesserung der Arbeit in Ihrem Projekt, zum Beispiel durch eine → Retrospektive.

Ziel ist nicht gleich Utopie, ein Ziel ist *SMART (Specific, Measurable, Attainable, Relevant, Timely)* (vgl. [Der06]). Ziele, die diese Kriterien erfüllen, sind klarer und können leichter auf ihren Erfüllungsgrad geprüft werden als Ziele, die nicht SMART formuliert sind. Ungenaue Ziele bieten Raum für Missverständnisse. Achten Sie jedoch darauf, bei der Messung des Erfüllungsgrads Ihrer Ziele nicht nur Daten, sondern auch → Information zu erzeugen. Ein passender Rahmen dafür kann eine → Retrospektive sein. Und stören Sie sich nicht daran, wenn Sie nicht direkt im ersten Wurf perfekte Ziele für Ihr Projekt definieren: Auch der Weg kann das Ziel sein – wenn er SMART ist. ■

Ein „Lexikon der Missverständnisse“ hat im Redaktionsteam kontroverse Diskussionen ausgelöst:

- Was ist der → Value dieses Beitrags für die praktische Arbeit eines → Teams?
- Was können die → User (in unserem Fall die Leser von OBJEKTspektrum) für ihre Arbeit lernen?
- Brauchen wir nicht einen roten Faden für die Einträge im Lexikon, um eine → Story (ach nein, die fehlt im Lexikon zugunsten der → Selbstorganisation) daraus zu machen?
- ...

Man kann diese Liste noch länger fortsetzen. Die endgültige Entscheidung, den Artikel trotz dieser Bedenken in das Heft zu nehmen, fiel während der persönlichen → Retrospektive auf die drei Tage „OBJEKTspektrum Information Days“ zum Thema → Agilität als Strategie, während derer wirklich gute Vorträge über Erfahrungen von erfolgreichen (und auch weniger erfolgreichen) Beispielen und von Einführungen agiler Organisationen gehalten und diskutiert wurden. Neben ganz viel inhaltlicher Substanz sind mir dabei auch ganz viele evangelistische Worthülsen im Gedächtnis geblieben. Spätestens → jetzt war mir klar: Wer dieses Lexikon aufmerksam liest, dem hilft es, manchen Phrasendrescher der agilen Szene zu entlarven. Und damit wäre ja schon ein wertvolles → Ziel erreicht. → Yes!

Thorsten Janning, Chefredakteur OBJEKTspektrum.

Gedanken der Redaktion zum Artikel von Thomas Epping.