mehr zum thema:
www.scrum.org

SCHÄTZEN IM DREIWOCHENTAKT: ERFAHRUNGEN AUS ÜBER 200 STORYS

Aufwandsschätzungen in Softwareprojekten sind eine Herausforderung. Auch agile Prozesse wie Scrum bewirken hier keine Wunder. Die kurzen Iterationen von Scrum liefern allerdings schnell Feedback zur Schätzgenauigkeit. In diesem Artikel beschreiben wir unsere Analyse von über 200 Storys hinsichtlich der Ursachen von groben Schätzfehlern und die daraus abgeleiteten Maßnahmen. Diese Maßnahmen verbesserten nicht nur die Schätzgenauigkeit, sondern führten auch zu einer Produktivitätssteigerung des Teams.

die autoren

Dr. Frank Beeh
(E-Mail: Frank.Beeh@zuehlke.com)
arbeitet als Softwarearchitekt bei der Firma Zühlke. Er ist begeisterter Anhänger agiler Methoden und begleitet als Certified ScrumMaster Kunden bei der Einführung von Scrum.

Dr. Raimond Reichert
(E-Mail: raimond.reichert@swisscom.com)
ist technischer Produktmanager bei Swisscom. Er engagiert sich für nachhaltige Veränderungsprozesse und sorgt dafür, dass die Mitarbeiter optimal einbezogen werden.

Umfeld und Vorgehen

Unser Team betreut eine Web-Applikation einer Telekommunikations-Firma für die Erfassung komplexer Aufträge durch den Kunden. Die Applikation erfasst die Daten von verschiedenen Systemen, um die für die Bestellung notwendigen Informationen zu sammeln und schließlich zusammen mit den Eingaben des Kunden als Bestellauftrag an ein Verarbeitungssystem zu schicken. Wir arbeiten im Maintenance-Modus: In vierteljährlichen Releases werden neue Funktionen hinzugefügt – beispielsweise neue Produkte oder Anpassungen an Abläufe. Bei jedem Release werden mehrere Projekte gleichzeitig realisiert und von einem circa zehnköpfigen Entwicklerteam umgesetzt.

Vor der Einführung von Scrum wurden einzelne Entwickler den Projekten fest zugeordnet. Das führte dazu, dass es kaum Möglichkeiten zur Zusammenarbeit gab, obwohl alle gleichzeitig an derselben Codebasis arbeiteten. Inzwischen wird teamintern Scrum als Entwicklungsprozess eingesetzt. Unverändert ist dabei allerdings die Einbindung in einen übergeordneten Wasserfall-Prozess, der pro Release die Entwicklung an allen betroffenen Systemen koordiniert. Durch die Organisation des Unternehmens ist eine strikte Trennung der einzelnen Rollen in unterschiedliche Teams vorgeben. Unser Team besteht aus Entwicklern, während Analysten und Tester anderen Teams zugeordnet sind. Zusammen führen diese organisatorischen Randbedingungen zu Einschränkungen in der Umsetzung von Scrum – so genannten ScrumButs – auf die wir im Folgenden weiter eingehen werden.

Abbildung 1 zeigt das Umfeld aus unserer entwicklerzentrierten Perspektive (Nr. 02): Business-Analysten erhalten Anforderungen von diversen Business-Stakeholdern und erstellen daraus Spezifikationen für die Entwickler (Schnittstellen mit Nr. 01) und Tester (Schnittstelle Nr. 05). Da die Applikation Teil einer stark verteilten Systemlandschaft ist, arbeiten unsere Entwickler eng mit den Entwicklern diverser Drittsysteme zusammen, die wiederum mit den Entwicklern anderer Drittsysteme zusammenarbeiten (Schnittstellen mit Nr. 03). Die Entwickler übergeben schließlich die Software den Testern (Nr. 06). Die Infrastruktur (Nr. 04) wird ebenfalls größtenteils extern betrieben.

Abb. 1: Umfeld des Entwicklerteams.

Der *Product Owner (PO)* sammelt die Anforderungen aller Projekte als Storys im *Product Backlog (PB)* und priorisiert sie, sodass eine eindeutige, projektübergreifende Bearbeitungsreihenfolge entsteht (siehe Kasten 1). Diese Storys werden vom gesamten Team mit Hilfe von Planungspoker in *Story Points (SPs)* (vgl. [Coh05]) geschätzt. Die Größe einer Story ist hierbei so gewählt, dass sie in einem bis maximal fünf Arbeitstagen durch einen oder mehrere Entwickler umgesetzt werden kann. Die Realisierung findet in Iterationen – so genannten *Sprints* – statt. Zu Beginn jedes Sprints legt der PO zunächst ein Sprintziel fest, das in der Regel Teile von mehreren Projekten umfasst.

Wir empfehlen, die Arbeiten von Teams, die an unterschiedlichen, parallel geführten Projekten auf der gleichen Codebasis arbeiten, in einem *Product Backlog (PB)* zusammenzuführen. Nur so ist die Basis für eine Zusammenarbeit als Team gegeben, ohne die eine Einführung von Scrum nicht sinnvoll ist. Der PO ist dabei besonders gefordert, da er die Ansprüche der verschiedenen Projekte unter einen Hut bringen und die Verantwortung für die projektübergreifende Priorisierung übernehmen muss.

Kasten 1: *Tipps für die Handhabung mehrerer parallel geführter Projekte.*

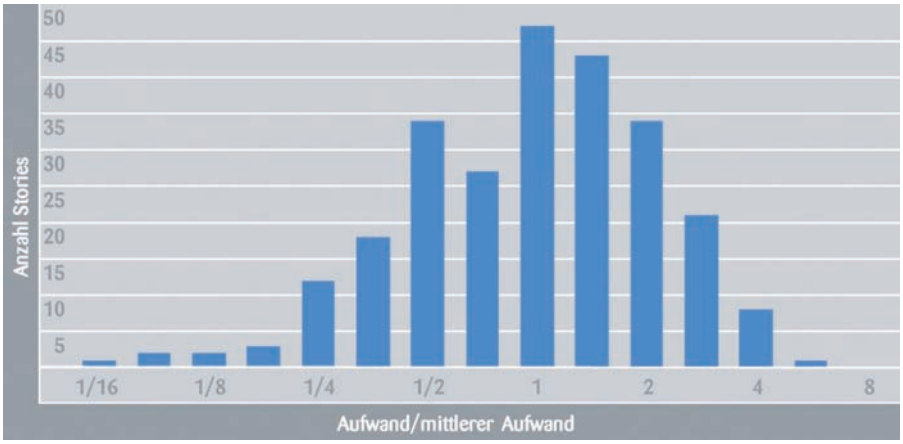

Abb. 2: *Histogramm über die Abweichung vom mittleren Aufwand.*

Beispielsweise kann ein Teilziel die Implementierung einer dynamischen Web-Seite für ein Projekt sein, damit diese den Stakeholdern beim Sprint-Review präsentiert werden kann. Für ein anderes Projekt ist die Anbindung eines neuen Web-Service inklusive *End-to-End*-Test auf einem Testsystem das zu erreichende Teilziel, damit dem Entwicklerteam des Web-Service Feedback gegeben werden kann.

Im ersten Teil der Sprint-Planung prüft das Team, wie die unterschiedlichen Teilziele so umgesetzt werden können, dass einerseits möglichst viele Synergien genutzt und andererseits Konflikte bei der Änderung benachbarter Bereiche im Quellcode minimiert werden. Basierend auf der zur Verfügung stehenden Kapazität entscheidet das Team, wie viele Storys in der Iteration umgesetzt werden können. Im zweiten Teil der Sprint-Planung werden die Storys in Tasks unterteilt. Hierbei arbeiten die Entwickler in Zweierteams. Zum Abschluss werden die erstellten Tasks gemeinsam geprüft, um bei Bedarf Ergänzungen und Korrekturen der Aufwandschätzungen vorzunehmen. Nach Fertigstellung der Sprint-Planung liegt eine initiale Gesamtstunden-Schätzung für das Sprint-Burndown vor.

Während des Sprints werden diese Schätzungen täglich aktualisiert. Zusätzlich wird der tatsächlich geleistete Aufwand erfasst, da diese Information für die Verrechnung mit den einzelnen Projekten benötigt wird. Das Vorhandensein dieser Daten ermöglicht uns den Vergleich der ursprünglichen Schätzung mit dem tatsächlichen Aufwand.

Auswertung der Schätzgenauigkeit

Nach der Fertigstellung einer Story liegt sowohl die ursprüngliche Größenschätzung in SPs als auch der tatsächlich geleistete Aufwand in Stunden vor. Damit die Daten der unterschiedlich großen Storys miteinander verglichen werden können, wird zunächst der geleistete Aufwand durch die Anzahl der SPs geteilt und man erhält den relativen Aufwand in h/SP.

Um ein Maß für die Schätzqualität der einzelnen Storys zu bekommen, müssen deren Schätzungen zunächst normiert werden. Hierzu verwenden wir den mittleren relativen Aufwand (ebenfalls in h/SP). Hierdurch erhält man für jede Story den Faktor, um wie viel ihr relativer Aufwand

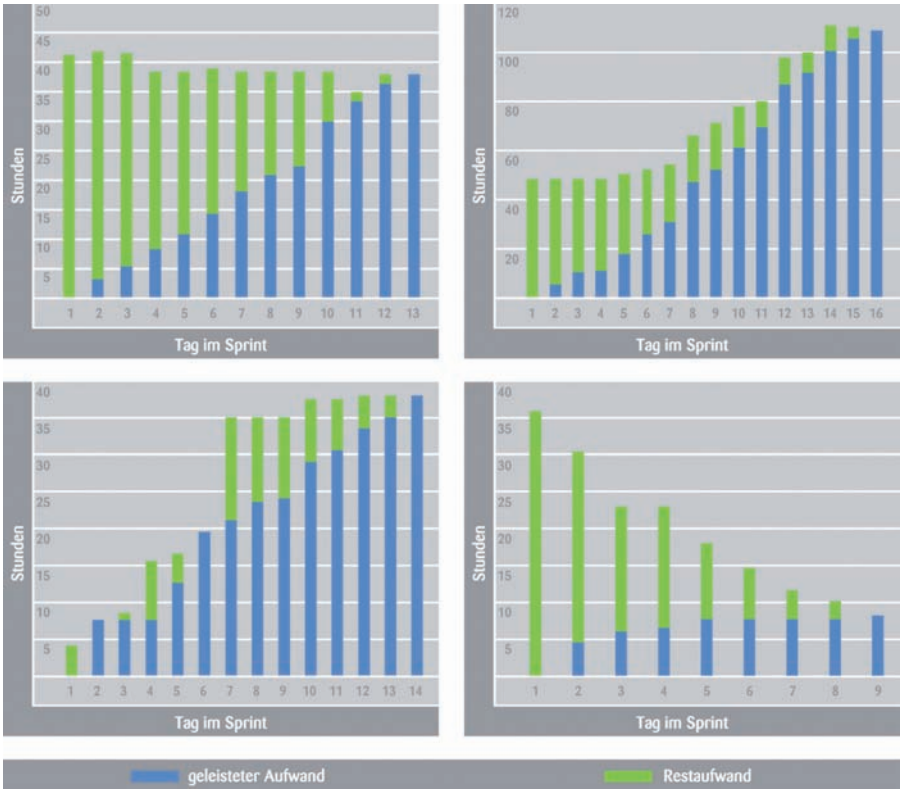

Abb. 3: *Zeitliche Verläufe von geschätztem Restaufwand und geleistetem Aufwand.*

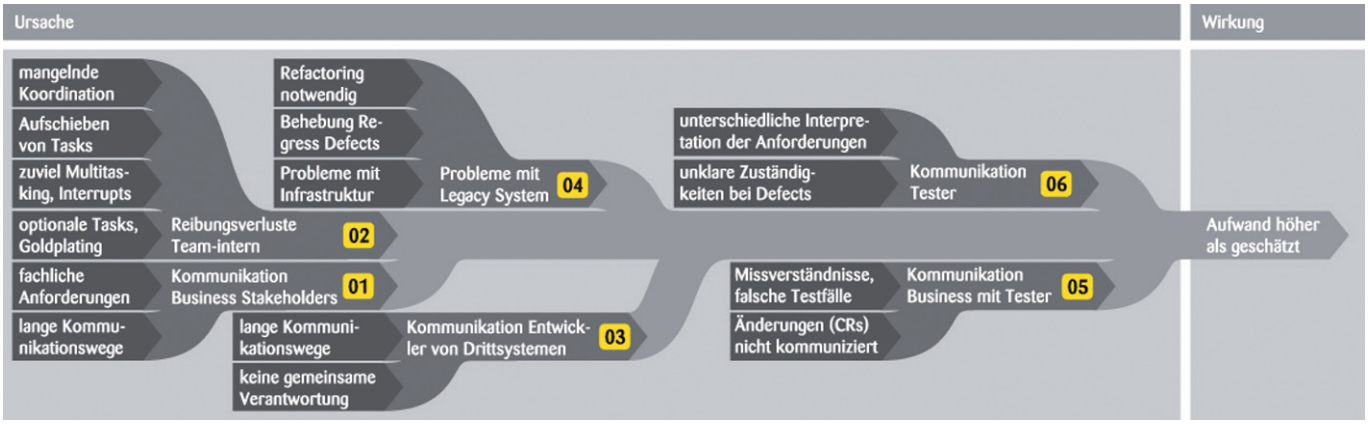

Abb. 4: Ursachenanalyse der Schätzfehler.

über oder unter dem Mittelwert liegt. Ein Faktor von 1 bedeutet, dass der Aufwand der Story dem Mittelwert entspricht. Die Faktoren 0,5 oder 2 stehen für einen halb oder doppelt so hohen Aufwand. Abbildung 2 zeigt das Histogramm der Faktoren aller Storys in einer logarithmischen Skala. Eine logarithmische Darstellung wurde in diesem Fall gewählt, weil diese die übliche Verteilung der Dauer eines Prozesses besser annähert. In unserem Fall ist erkennbar, dass sich eine gute Annäherung an eine logarithmische Normalverteilung ergibt. Dies zeigt, dass unsere Daten stochastisch aussagekräftig sind.

Abbildung 3 zeigt den geschätzten Restaufwand (grün) und den bisher geleisteten Aufwand (blau) im Verlauf der Tage eines Sprints für einige interessante Fälle. Links oben ist eine gut geschätzte Story zu sehen. Die initiale Schätzung stimmt gut mit dem tatsächlichen Aufwand überein. Rechts oben ist der Fall zu erkennen, bei dem der Aufwand immer mehr ansteigt und die Schätzungen zu zaghaft angepasst wurden. Links unten ist zunächst das gleiche Problem zu erkennen, bis schließlich eine gezielte Neuschätzung zu einer erheblich beständigeren Schätzung führt. Eine Überschätzung des Aufwandes ist selbstverständlich auch möglich. Rechts unten in Abbildung 3 ist ein Beispiel hierfür zu sehen.

Warum haben wir Storys massiv unterschätzt?

Einzelne Storys haben wir stark unterschätzt. Es hatte den Anschein, als handelte es sich um ganz bestimmte Arten von Storys. So wurde es schon fast zum „Running Gag", dass Rabatte und Discounts in den von uns entwickelten Bestellprozessen in der Entwicklung immer viel teurer waren als gedacht. Die Arbeit an unserer Infrastruktur (Testsystem, Entwicklungsumgebung inklusive kontinuierlicher Integration) dauerte ebenfalls meist länger als geschätzt.

Wir unterziehen die massiv unterschätzten Storys immer wieder im Rahmen der Retrospektive einer Fehler-Ursachen-Analyse (vgl. [Wik-a]) und versuchen, den zu Grunde liegenden Problemen näher zu kommen. Dabei stehen für uns die konkreten Ursachen im Vordergrund, zu denen wir auch Maßnahmen entwickeln können. Hierbei ist es wichtig, die Ursachen primär bei uns selbst zu suchen und nicht nur externe Einflüsse verantwortlich zu machen. Die Versuchung ist allerdings groß, die Verantwortung von sich zu weisen. Die typische Fallunterscheidung der Ursachen von Erfolg und Misserfolg („Für meine Erfolge bin ich selbst verantwortlich, für meine Misserfolge externe Faktoren", vgl. [Wik-b]) gilt auch für Aufwandschätzungen: Für gute Schätzungen übernimmt man gerne die Verantwortung und weist etwa auf die eigene Erfahrung hin, aber an schlechten Schätzungen sind Faktoren außerhalb der eigenen Kontrolle schuld (vgl. [Jor04]). Dieses Denkschema ist für eine effektive Retrospektive nicht hilfreich. Abbildung 4 zeigt die von uns ermittelten Ursachen der Schätzfehler als Fischgräten-Diagramm. Der Übersichtlichkeit halber sind nur die ersten zwei Ebenen der Ursachenanalyse dargestellt. Die Nummerierung ordnet die Ursachen dem Kontext zu (siehe Abbildung 1). Basierend auf diesen Problemursachen haben wir unsere Maßnahmen erstellt. Sie fließen in unsere Vorsätze für Entwickler, ScrumMaster und PO ein, die in der täglichen Arbeit befolgt werden (siehe Kasten 2).

Die Entwickler:
- arbeiten mit an der Spezifikation
- kommunizieren Zusatzaufwand proaktiv
- übernehmen explizit Verantwortung für eine Story
- arbeiten mindestens zu zweit an einer Story
- vermeiden gegenseitige Unterbrechungen
- melden teamübergreifende Hindernisse frühzeitig
- machen teamübergreifendes Pair Programming
- überprüfen gemeinsam die Aufwandschätzungen

Der ScrumMaster:
- organisiert teamübergreifende Sprint-Reviews
- organisiert teamübergreifende Retrospektiven
- sorgt für eine zeitnahe Analyse von Storys mit Mehraufwand

Kasten 2: Neu beschlossene Maßnahmen des Teams zur Verbesserung der Schätzgenauigkeit.

Ursache 1: Anforderungserhebung

Bei einigen Storys ist die Schätzung zu niedrig, weil die Entwickler die fachliche Komplexität der Anforderungen nicht erkannt oder Fehler in der Spezifikation (z. B. Widersprüche und Lücken) übersehen haben. Da unsere Projekte Teile von über-

geordneten Wasserfall-Projekten sind, werden uns Spezifikationen übergeben, die als abgeschlossen betrachtet werden. Diese Übergabe bringt einen großen Reibungsverlust mit sich: Den Entwicklern fehlt der Kontext der Anforderungsentstehung und sie müssen sich für die Schätzung in kürzester Zeit in die Anforderung hineindenken. Im Sinne von Scrum und der Idee der interdisziplinären Teams fordern wir daher eine engere Mitarbeit der Entwickler während der Spezifikationsphase.

Maßnahme: Die Entwickler unterstützen die Business-Analysten bereits bei der Erstellung der Anforderungen. Der ScrumMaster stellt sicher, dass die Entwickler vor der initialen Aufwandschätzung in SPs zudem genügend Zeit für ein gründliches Review der Anforderungen haben. Die für die Unterstützung und das Review benötigte Zeit wird von den Entwicklern selbst geschätzt; entsprechende Tasks werden ins *Sprint Backlog* aufgenommen.

Bei der Umsetzung tauchen häufig zusätzliche Anforderungen auf. Dies können neue fachliche Anforderungen oder Änderungsanforderungen sein, die bereits entwickelt werden. Zudem werden Anforderungen während der Entwicklung naturgemäß gründlicher durchdacht als zuvor, sodass weitere Unvollständigkeiten und Widersprüche zu Tage treten, die Mehraufwand verursachen. Diese Ursachen für zusätzliche Anforderungen lassen sich kaum vermeiden. Es stellt sich die Frage, wie sich das Team in solchen Fällen verhalten soll. Da das Sprint-Ziel sich aus den Sprint-Zielen mehrerer Projekte zusammensetzt, brauchen wir eine Art Frühwarnsystem, um rechtzeitig reagieren zu können, sobald einzelne Storys eines Projekts die Sprint-Ziele anderer Projekte gefährden.

Maßnahme: Wenn ein Entwickler am Daily Meeting Zusatzaufwand meldet, entscheidet das Team unmittelbar aktiv, wie das weitere Vorgehen zu der entsprechenden Story sein soll: Kann der Mehraufwand geleistet werden, ohne die Sprint-Ziele zu gefährden? Braucht es eine Neuplanung mit dem PO, wobei zum Beispiel entschieden wird, ob die Story aufgeteilt und eine Folgestory für den nächsten Sprint erfasst wird?

Diese Maßnahme wird auch für alle anderen Ursachen eingesetzt, die zu einem Mehraufwand führen.

Ursache 2: Teaminterne Reibungsverluste

Die Arbeitszuteilung geschieht nach Scrum selbstorganisierend: Die Entwickler entscheiden selbst, welchen Task sie als nächstes bearbeiten (unter Berücksichtigung der Priorität der Storys). An größeren Storys arbeiten zwei bis vier Entwickler gemeinsam. Dabei kann es schon einmal vorkommen, dass sie unbeabsichtigt gegeneinander statt miteinander arbeiten, wenn zum Beispiel verschiedene Entwickler das gleiche Problem lösen. Kommen sich die Entwickler im Code zu nahe, entstehen unnötige Konflikte, deren Auflösung zeitraubend sein kann. Solche Konflikte lassen sich auch mit *Pair Programming* nicht vermeiden, wenn mehr als ein Entwicklerpaar an einer Story arbeitet. Da wir zudem in einem Sprint Ziele mehrerer Projekte umsetzen, entstehen Konflikte auch dadurch, dass verschiedene Projekte Anforderungen stellen, die die gleichen Codebereiche betreffen. Die Problemursache in diesem Fall ist mangelnde Koordination unter den Entwicklern.

Maßnahme: Ein Entwickler übernimmt die Verantwortung für eine Story und koordiniert die Implementierung der Story mit den anderen Entwicklern.

„Was du heute kannst besorgen, das verschiebe doch auf übermorgen!" Das Aufschieben von unangenehmer Arbeit (auf neudeutsch „Prokrastination") hat viele Ursachen (vgl. [Wik-c]). Unserer Erfahrung nach bleiben unbeliebte Storys gerne liegen. Das sind zum Beispiel Storys, die Altlasten im Code betreffen oder aufwändige Abklärungen benötigen. Solche Storys überlässt man dankbar einem anderen Entwickler und sucht sich eine angenehmere. Das führt dazu, dass sich die Umsetzung dieser Storys unnötig in die Länge zieht. Ist der Entwickler abwesend, bleibt die Story ungeachtet ihrer Priorität unbearbeitet. Zudem werden Abklärungen mit Drittpersonen außerhalb des Entwicklerteams oft nicht genügend proaktiv vorangetrieben.

Maßnahme: An einer Story arbeiten mindestens zwei Entwickler. Diesen Vorsatz hatten wir bereits früher gefasst, um den Know-how-Transfer sicherzustellen. Zusätzlich stellt diese Maßnahme sicher, dass sich zwei Leute für eine Story verantwortlich fühlen und bei Schwierigkeiten gegenseitig unterstützen.

Maßnahme: Im Daily Meeting werden auch die Storys besprochen, die zwar in Arbeit sind, an denen aber am Vortag niemand gearbeitet hat und die daher keine Erwähnung finden. Das Team entscheidet, wie das weitere Vorgehen zu der entsprechenden Story sein soll: Ist sie tatsächlich mit gutem Grund in der Warteschleife oder kann die Umsetzung vorangetrieben werden?

Arbeitet ein Entwickler an vielen verschiedenen Tasks gleichzeitig, muss er häufig den Kontext wechseln. Zudem steigt mit der Anzahl der Tasks in der Regel auch der Kommunikationsaufwand und damit die Anzahl der Unterbrechungen. Kontextwechsel und Unterbrechungen wiederum bedeuten Reibungsverlust – die effektive Produktivität sinkt.

Maßnahme: Das Team nimmt sich vor, gegenseitige Unterbrechungen möglichst zu vermeiden. Benötigt ein Entwickler die Unterstützung von einem anderen Entwickler, schickt er ihm per Mail eine Anfrage. Der andere Entwickler wird so nicht in der Arbeit gestört und kann hierauf reagieren, sobald es für ihn günstig ist. Diese Regel gilt auch für alle Personen außerhalb des Teams. Für dringende Anfragen steht ihnen zusätzlich jederzeit der PO zur Verfügung.

Ursache 3: Kommunikation mit Entwicklern von Drittsystemen

Storys mit Abhängigkeiten zu Drittsystemen sind oft schwierig zu schätzen. So können die für die Schätzung getroffenen Annahmen über die Komplexität der Schnittstelle falsch sein. Entsprechend verursachen die Abklärungen zu den Schnittstellen während der Implementierung Mehraufwand. Häufig befinden sich die Entwickler von Drittsystemen an anderen Standorten und kennen sich nicht persönlich – die geographische Distanz erschwert die Zusammenarbeit zusätzlich. Weiterhin werden die Entwickler durch die Nichteinhaltung von vereinbarten Lieferterminen oder die schlechte Verfügbarkeit von Drittsystemen ausgebremst. Unsere Erfahrung zeigt, dass die einzelnen Entwickler oft Mühe haben, solchen organisationseinheitsübergreifende Hindernisse aus dem Weg zu räumen.

Maßnahme: Die Entwickler vereinbaren zunächst selbständig Liefertermine mit den Drittsystemen. Sofern diese dann nicht eingehalten werden oder die Systeme schlecht verfügbar sind, melden sie dies frühzeitig

als Hindernis an den ScrumMaster gemäß der Devise: Eskalieren, nicht resignieren. Der Scrum Master unterstützt die Entwickler bei der Beseitigung der Hindernisse. Bei einer Gefährdung des Sprint-Ziels entscheidet der PO über das weitere Vorgehen.

Maßnahme: Die Entwickler nutzen die Möglichkeit zum Pair Programming mit Entwicklern von Drittsystemen, um etwa Schnittstellen gemeinsam zu erarbeiten. Falls nötig, wird der hierfür benötigte Reiseaufwand in Kauf genommen.

Ursache 4: Probleme mit dem Legacy-System

Legacy-Systeme bergen häufig Überraschungen, die bei Schätzungen nur schwer zu berücksichtigen sind: Es kann sein, dass vor der Umsetzung einer Anforderung eine nicht geplante Refaktorisierung notwendig ist. Es können Fehler auftauchen, die schon lange im System sind, aber erst jetzt entdeckt werden. Die technische Infrastruktur kann komplizierter sein als angenommen. Das Hauptproblem ist meist, dass das notwendige Detailwissen fehlt, weil die damaligen Entwickler das Team bereits verlassen haben.

Bei der Auswertung der massiv unterschätzten Storys fällt auf, dass bei ihnen der relative Aufwand deutlich über der Norm liegt. Die Detailschätzung ist pessimistischer als die erste Grobschätzung in SPs. Das sollte ein Warnsignal sein: Vielleicht haben wir bei der SP-Schätzung etwas übersehen? Unsere Erfahrung zeigt, dass das Risiko bei solchen Storys besonders hoch ist, da die Detailschätzung immer noch zu optimistisch ist. Eine Beeinflussung der Entwickler durch die SP-Schätzung ist hier naheliegend.

Maßnahme: Im zweiten Teil der Sprint-Planung erstellen die Entwickler Tasks, die in Stunden geschätzt werden. Diese Stundenschätzung vergleichen sie mit der ursprünglichen SP-Schätzung und dem Erfahrungswert von Stunden pro SP. Liegt die Stundenschätzung deutlich über dem zu erwartenden Aufwand, werden die Tasks und ihre Schätzungen vom Team nochmals gründlich überprüft.

Die Durchführung eines Sprint-Reviews mit allen Stakeholdern bietet eine gute Möglichkeit, die in einem Wasserfall-Prozess eigentlich fest vorgegebenen Anforderungen diskutierbar zu machen: Mit dem Review der lauffähigen Software kann den Stakeholdern frühzeitig und lange vor Projektende anschaulich aufgezeigt werden, dass ein striktes Festhalten an Anforderungen dem Projekterfolg abträglich sein kann. Das Sprint-Review kann der Auslöser sein, um Anforderungen neu zu priorisieren, bestimmte Anforderungen auszubauen oder zu streichen. Damit kann die für Scrum notwendige Flexibilität in den Anforderungen erreicht werden: So wird es möglich, die agilen Werte „Customer collaboration over contract negotiation" und „Responding to change over following a plan" (vgl. [Bec01]) auch in einer ScrumBut-Organisation zu leben.

Kasten 3: Der spezielle Nutzen von Sprint-Reviews in nach Rollen organisierten Unternehmen.

Ursachen 5 und 6: Kommunikation mit Testern

Eine spezielle Herausforderung ist die team- und organisationseinheitenübergreifende Zusammenarbeit. In unserem Fall gehören Business-Analysten, Software-Ingenieure und Tester verschiedenen Organisationseinheiten an, die auf zwei Standorte verteilt sind. Das heißt, unser Team arbeitet zwar nach Scrum, aber nicht in einem interdisziplinären Team. Organisatorische Einschränkungen dieser Art sind unter dem Begriff *ScrumBut* bekannt geworden. Abbildung 1 zeigt unsere und auch anderweitig weit verbreitete ScrumBut-Organisationsform, während in Abbildung 5 als Kontrast dargestellt ist, wie unser Umfeld mit einem interdisziplinären Team aussehen würde.

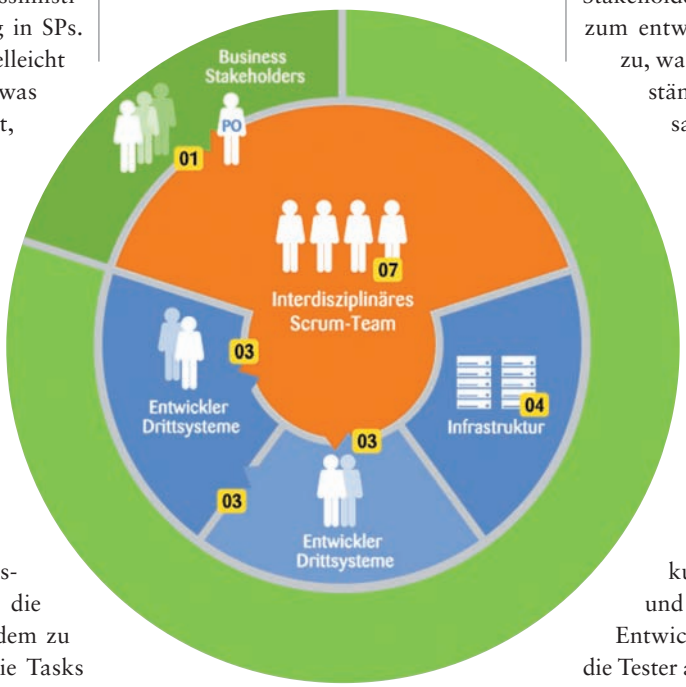

Abb. 5: Von Scrum vorgesehene Organisationsform.

Da wir die Organisationsform nicht so schnell ändern können, arbeiten wir mit *ScrumBut Workarounds*. Bei den Sprint-Reviews wird am Ende eines Sprints allen Stakeholdern der aktuelle Stand der Software und der umgesetzten Anforderungen vorgeführt: Vertretern des Business, Business-Analysten, Testern und Leitern der Projekte, an denen wir Teil haben. In unserer Organisation ist das Sprint-Review der erste Anlass, bei dem *alle* Stakeholder – insbesondere auch über die Grenzen von Organisationseinheiten hinweg – gemeinsam über die umgesetzten Anforderungen diskutieren konnten (siehe Kasten 3). Das gab es vor der Einführung von Scrum nicht und das Echo ist überwältigend positiv. Die Nähe der verschiedenen Stakeholder zu den Entwicklern und auch zum entwickelten System nahm deutlich zu, was wiederum das gegenseitige Verständnis und damit die künftige Zusammenarbeit verbesserte.

Maßnahme: Sprint-Reviews sind fester Bestandteil von Scrum. In einer ScrumBut-Organisationsform lohnt es sich, dazu alle Stakeholder einzuladen, insbesondere auch diejenigen, mit denen die Entwickler aus organisatorischen Gründen im Alltag sonst keinen Kontakt haben.

Dennoch gibt es viele Reibungspunkte in der Kommunikation: Die intensiven Diskussionen zwischen Entwicklern und Business-Analysten während der Entwicklung führen häufig dazu, dass die Tester am Ende nicht wissen, gegen welche Anforderung sie testen müssen. Bei häufigen Deployments auf Testumgebungen wissen die Tester manchmal nicht, wel-

che Anforderungen in der aktuellen Version umgesetzt sind und welche noch nicht.

Maßnahme: Um hier die Zusammenarbeit zu verbessern, werden teamübergreifende Retrospektiven durchgeführt. Diese Retrospektiven werden bei Bedarf gemacht und entsprechend den Bedürfnissen der jeweiligen Situation gestaltet: Teamübergreifende Maßnahmen erfordern viel Fingerspitzengefühl bei der Moderation durch den ScrumMaster.

Und last but not least braucht es noch eine Meta-Maßnahme: Die Analyse der deutlich unterschätzten Storys ist umso schwieriger, je länger ihre Umsetzung schon zurückliegt.

Meta-Maßnahme: Storys mit deutlichem Mehraufwand werden bei der nächsten Sprint-Retrospektive analysiert.

Damit schließt sich der Kreis von der initialen Schätzung bis hin zur Auswertung. Das Team kann neue Maßnahmen bereits im nächsten Sprint anwenden und hat eine Gelegenheit mehr, seine Arbeitsweise zu verbessern und die Effektivität weiter zu steigern.

Fazit

Unsere Erfahrung und die Auswertung der über 200 Storys bestätigen einmal mehr die PeopleWare-Maxime von Tom DeMarco, dass die größten Probleme in unserer Arbeit nicht technologischer, sondern soziologischer Natur sind. All unsere Maßnahmen betreffen die Organisation der Zusammenarbeit und die Übernahme von Verantwortung. Innerhalb des eigenen Teams hat man im Idealfall genügend Spielraum, damit die beschlossenen Maßnahmen greifen können. Wesentlich schwieriger wird es bei teamübergreifenden Maßnahmen, bei denen deutlich mehr Stakeholder betroffen sind und eine größere Bandbreite von potenziell in Konflikt stehenden Interessen berücksichtigt werden müssen. ■

Literatur & Links

[BecO1] K. Beck et al, Manifesto for Agile Software Development, 2001, siehe: www.agilemanifesto.org

[CohO5] M. Cohn: Agile Estimating and Planning, Addison-Wesley 2005

[JorO4] M. Jorgensen, K. Molokken-Ostvold, Reasons for Software Effort Estimation Error: Impact of Respondent Role, Information Collection Approach, and Data Analysis Method, in: Proc. of IEEE Transactions on Software Engineering, 2004

[Wik-a] Wikipedia, Fehler-Ursachen-Analyse, siehe: de.wikipedia.org/wiki/Fehler-Ursachen-Analyse

[Wik-b] Wikipedia, Kausalattribuierung, siehe: de.wikipedia.org/wiki/Kausalattribuierung

[Wik-c] Wikipedia, Aufschieben, siehe: de.wikipedia.org/wiki/Aufschieben