mehr zum thema:
infos.seibert-media.net/display/Websoftware/Agile+Vorhersagen

AGILES SCHÄTZEN IM TEAM: VERFAHREN IN DER AGILEN SOFTWAREENTWICKLUNG

Mit Aufwandsschätzungen machen es sich Software-Entwicklungsteams zu Recht schwer: Der Kunde will vorweg wissen, was ein neues Feature in seiner Software kostet. Selbst wenn wir den genannten Betrag (Aufwand) explizit als Schätzung deklarieren, wissen wir doch, was passieren wird: Im Nachhinein nagelt uns der Kunde darauf fest. Das versucht man in der agilen Softwareentwicklung zu umgehen und beschätzt deshalb keine Aufwände, sondern die Komplexität eines Features in Relation zu den anderen. Dieser Artikel beschreibt, wie agile Teams schnell zu solchen Schätzungen kommen.

▶ der autor

Joachim Seibert
(jseibert@seibert-media.net)
ist CTO und Geschäftsführer der //SEIBERT/MEDIA GmbH aus Wiesbaden. Als überzeugter Verfechter agiler Methodik und zertifizierter Scrum Professional hat er es sich zur Aufgabe gemacht, Teams und Unternehmen agiler zu machen.

Als Softwareentwickler wird man häufig mit der Frage konfrontiert: „Wie lange brauchst du dafür? Schätz doch mal!" Denn es ist ja ein durchaus verständliches Bedürfnis des Kunden oder Projektverantwortlichen zu wissen, wann die neue Funktion endlich fertig ist (Zeitplanung) und was das Ganze eigentlich kosten soll (Aufwand bzw. Kosten). Der Softwareentwickler steht nun aber vor einem Dilemma: Er kann und will nicht genau sagen, wie viele Stunden er noch benötigt, denn er vermag den exakten Verlauf seiner Arbeit nicht zu prognostizieren. Klappt alles wie geplant? Gibt es Probleme mit dieser geplanten SQL-Abfrage oder sieht die Webseite im Internet Explorer am Ende auch wirklich genauso aus wie in Firefox & Co.? Er kann nicht alle Eventualitäten absehen bzw. einkalkulieren.

Nun könnte der Kunde anregen: „Er kann doch eine Dreipunktschätzung abgeben und mir den mindesten, maximalen und den erwarteten Aufwand mitteilen." Damit hat sicher jeder schon mal gearbeitet. Dennoch: Abgesehen davon, dass zwischen den genannten Minimal- und Maximalangaben manchmal Welten liegen, gibt es auch keine Garantie dafür, dass der Höchstbetrag nicht überschritten wird. Es ist eben am Ende doch nur eine Schätzung. Wenn also Schätzungen letztlich sehr ungenau ausfallen, stellen sich folgende Fragen: Wie viel wollen wir in eine Vorabschätzung der Aufwände investieren? Soll der Entwickler sich richtig viel Zeit nehmen und versuchen, jedes geplante Feature auf die einzelnen umzusetzenden Bestandteile herunterzubrechen? Kann er das vorab überhaupt? Welche Aufwände müssen denn eigentlich in der Schätzung berücksichtigt werden? Nur der Implementierungsaufwand für die zu schätzende Funktion? Was ist mit Designern, Testern, Systemadministratoren und Projektmanagement (bzw. *Product Owner (PO)* und *ScrumMaster*), die an der Realisierung auch in gewisser Weise beteiligt sind?

Unter dem Motto „Unsere Schätzungen müssen besser werden" investieren Projektmanager und ihre Softwareentwickler teilweise richtig viel Zeit und entwickeln komplizierte Schätzverfahren, um vermeintlich genauere Schätzungen zu generieren. So werden Vergangenheitswerte betrachtet und Risikoaufschläge optimiert. Man diskutiert, wie hoch ein Projektmanagement- und Testaufschlag sein müsste. Dabei ist die „genaue Schätzung" doch ein Widerspruch in sich. Wenn wir es genauer wüssten, dann müssten wir nicht schätzen. Ein Mitarbeiter von 1&1 hat auf die Frage nach der Genauigkeit einer Schätzung und dem Zeitaufwand, um sie zu erhalten, eine einfache und konsequente Antwort parat: den selbst gebastelten Schätzwürfel (vgl. [Sch], siehe Abbildung 1). Wenn wir sowieso nicht genau wissen, was auf uns zukommt, wieso dann nicht einfach würfeln?

Ich gebe zu: Diese Methode ist auch mir nicht wissenschaftlich genug. Schließlich ist Softwareentwicklung kein Glücksspiel. Aber haben wir denn Alternativen, die mit vertretbarem Aufwand einhergehen? Wie wäre es, wenn wir – statt zu würfeln – es etwas genauer versuchen und wenigstens die geplanten Features relativ zueinander nach ihrer Komplexität bewerten? Das ist eine Aussage, die dem Kunden sicher hilft: Wie teuer wird wohl welches Feature im Verhältnis zu einem anderen – ohne zu diesem Zeitpunkt einen konkreten Euro-Betrag direkt schon nennen zu können.

Relatives Schätzen: Relationen herstellen

„Dieses ist größer als jenes." Aussagen dieser Art treffen wir im Alltag häufig, teilweise auch, ohne konkrete Werte zu nennen. Die Übung in Kasten 1 soll ein solches

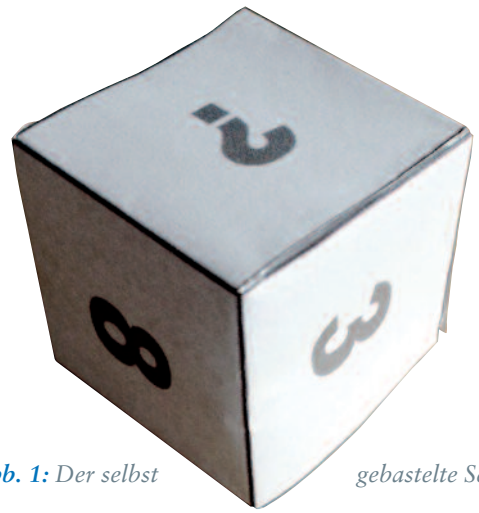

Abb. 1: Der selbst gebastelte Schätzwürfel.

Was ist relatives Schätzen? Dazu ein Alltagsbeispiel fernab der Softwareentwicklung: Mich fragt jemand, ob ich denn wisse, wie groß die Fläche folgenden Länder in Quadratkilometern ist: USA, Deutschland (DE), Schweiz (CH) und Frankreich (FR).
Kurz gesagt: Ich habe keine Ahnung, Wikipedia weiß es sicher – ich nicht. Allerdings kann ich auf jeden Fall sagen, welches Land flächenmäßig größer als das andere ist. Ich kann also eine Größen-Reihenfolge herstellen: *CH < DE < FR < USA*.
Wenn es noch etwas genauer sein soll, kann ich zusätzlich versuchen, die Größenrelation der Länder in Zahlenwerten zu schätzen: *CH: 1 < DE: 5 < FR: 8 < USA: 100*.
Eine Recherche bei Wikipedia ergibt nun: *CH: 41 Tkm² < DE: 357 Tkm² < FR: 547 Tkm² < USA: 9.809 Tkm²*.
In Verhältnisangaben übersetzt, heißt das: *CH: 0,6 < DE: 5 < FR: 7,7 < USA: 137*

Gar nicht schlecht getroffen, oder? Wie ist diese Schätzung zustande gekommen?

1. Zuallererst wird ein Referenzwert gesucht, den man gut kennt und abschätzen kann. In diesem Fall ist das für mich natürlich Deutschland.
2. Für diese Referenzgröße wird nun (scheinbar) willkürlich ein Referenzwert vergeben, hier z. B. der Wert 5.
3. Davon ausgehend, werden die anderen Länder bzw. deren relatives Verhältnis zueinander geschätzt. Das fällt leicht bei Dingen mit ähnlicher Größe, hier CH, DE, FR.
4. Bei verhältnismäßig großen Dingen fällt uns die Beschätzung allerdings schwer und wird ungenau, wie das Beispiel USA zeigt.

Kasten 1: *Relatives Schätzen: ein Alltagsbeispiel.*

Alltagsbeispiel aufzeigen. Als Softwareentwickler schätzen wir nun natürlich nicht so triviale Dinge wie die Größen von Ländern, die man am Ende ja auch (Wikipedia sei Dank) einfach nachschlagen kann. Teilweise wünscht sich der Kunde für seine Software relativ große neue Features, deren einzelne Facetten im Vorfeld gar nicht absehbar sind, geschweige denn die notwendigen Arbeitsschritte. Wie lange wir daran sitzen, welchen Aufwand dieses Feature also produziert, können wir zu diesem Zeitpunkt nicht sagen. Die Frage, wie komplex eine Anforderung im Verhältnis zu einer anderen ist, können wir allerdings beantworten.

Komplex bedeutet dabei erst einmal nicht, dass die Umsetzung besonders zeitintensiv und mit großem Aufwand verbunden ist. Komplexität ist vielschichtiger. Beispiele könnten sein:

- Es gibt einen komplizierten Ablauf (Workflow), der durchschritten werden soll.
- Es sind viele Bereiche der Software betroffen.
- Es sind sehr viele Änderungen vorzunehmen.
- Es sind viele Personen involviert.

Wenn uns der Kunde eine Liste mit gewünschten Features vorlegt, könnten wir also versuchen, diese (genau wie bei dem Länder-Beispiel in Kasten 1) nach ihrer Komplexität zu ordnen und in einem zweiten Schritt die Relationen zu beschätzen.

Fibonacci-Schätzskala

Sobald wir Verhältnisse schätzen, stellt sich die Frage der Skala. Beim Beschätzen der Länder haben wir gelernt, dass die Beschätzungen ungenauer werden, wenn die Dinge größer sind. Das sollte unsere Skala berücksichtigen.

Eine Zahlenfolge, die die Eigenschaft mitbringt, nach oben immer größere Abstände zu haben, ist die Fibonacci-Folge: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, … Daran angelehnt, hat sich die heute in der agilen Welt etablierte Skala entwickelt, die zur Beschätzung herangezogen wird: 1, 2, 3, 5, 8, 13, 20, 40, 100. Die wichtigste Eigenschaft dieser Zahlenfolge: Je größer die Zahl ist, desto größer ist der Abstand zur letzten – also genau das, was wir vorhin festgestellt haben: Je größer die Schätzung ist, desto ungenauer wird sie.

Die Crux der Story-Points

Die Zahlen bzw. Größen der Verhältnisangaben werden in der agilen Welt häufig *Story-Points* (von User-Story) oder Komplexitätspunkte genannt. Das hat sich etabliert, ist aber eigentlich missverständlich, denn es impliziert eine Werteinheit und eben kein Verhältnis mehr. Daraus entsteht dann häufig das Missverständnis, man könne doch gleich in Personentagen beschätzen bzw. direkt umrechnen, in der Art: „Ein Story-Point entspricht bei uns fünf Personentagen."

Natürlich sind Umrechnungen von Komplexität in Aufwand in gewissen Maße notwendig – schließlich möchte der Kunde abschätzen können, wie weit er den Geldbeutel für ein Feature aufmachen muss. Allerdings ist eine Umrechnung von Komplexitätsverhältnissen in Aufwand nicht trivial. Dazu später mehr.

Das Team schätzt

Wichtig in agilen Softwareprojekten ist die Autonomie bzw. die Selbstorganisation des Teams. Diese ist nur dann gewährleistet, wenn Entscheidungen auch im Team getroffen werden. Dazu zählt unter anderem auch die Aufwandsschätzung, ansonsten steckt gerade in diesen Schätzungen immer wieder Konfliktpotenzial: Einzelne fühlen sich oder vielmehr ihre Bedenken bezüglich der Beschätzung nicht berücksichtigt. „Habe ich doch gleich gewusst, dass diese Schätzung unrealistisch ist", ist eine typische Aussage von Entwicklern, die sich mit Schätzungen konfrontiert sehen, an denen sie nicht beteiligt waren.

Es schätzt also das Team. Das Management erwidert daraufhin häufig: „Das ist aber unheimlich teuer. Die verbringen doch auch so schon genug Zeit in Meetings." Noch ein Grund mehr, warum wir Verfahren brauchen, um möglichst schnell zu einer Schätzung zu kommen. Neben der gerade angeführten Akzeptanz sei diesem Manager als weiteres Argument genannt, dass es sehr wichtig ist, dass das Team sich frühzeitig mit den Anforderungen beschäftigt, damit alle wissen, was auf sie zukommt.

Teams in agilen Projekten sind optimalerweise *cross-functional* besetzt. Neben Softwareentwicklern gehören auch Designer, Tester, Systemadministratoren (auf neudeutsch: *DevOps*) zum Team. Wenn wir nun also keine Aufwände schätzen, sondern auf relative Komplexitätsabschätzungen hinarbeiten, hat das den großen Vorteil, dass alle im Team aktiv an dieser Schätzung teilnehmen können und so eine

- Jedes Teammitglied erhält einen Satz Karten, auf denen die Zahlen der abgewandelten Fibonacci-Folge stehen.
- Häufig gibt es zusätzliche Karten wie „?", „Unendlich" oder „Kaffeepause", um den Teilnehmern die Möglichkeit zu geben, mit den Karten ihre Bedürfnisse nach einer genaueren Erklärung oder einer Meetingpause zu visualisieren.
- Der PO stellt die erste Anforderung vor und beantwortet Fragen des Teams.
- Der ScrumMaster fragt das Team, ob es bereit zum Pokern ist, und auf „Drei, zwei, eins!" zeigt jeder Teilnehmer die von ihm ausgewählte Karte.
- Nun erklären sich derjenige mit der höchsten Schätzung und derjenige mit der niedrigsten gegenseitig, warum sie entsprechend geschätzt haben.
- Es erfolgt eine zweite Pokerrunde. Die kurze Variante: Nach der zweiten Runde wird stets die größte genannte Schätzung verwendet. Liegen die Schätzungen allerdings weiterhin sehr weit auseinander, sollte die Story zuerst „zwischengeparkt" und am Ende erneut betrachtet werden.
- Die ausführlichere Variante: Es werden so viele Pokerrunden durchgeführt, bis alle die gleiche Schätzung abgegeben haben.

Kasten 2: Planungspoker-Spielregeln.

tatsächliche Teamschätzung erreicht wird. Eine typische Aussage, wie „Ich weiß doch nicht, wie lange du für die Implementierung brauchst", ist somit obsolet.

Wie können wir aber sicherstellen, dass das Team sich in diesen Anforderungsworkshops (in Scrum auch häufig *Backlog-Grooming* oder auch explizit *Estimation Meeting* genannt) nicht in ellenlange Detaildiskussionen verstrickt, sondern diese Treffen so kurz und effizient wie nur möglich gestaltet? Dazu wäre eine feste Vorgehensweise sinnvoll, um möglichst rasch und reibungslos zur oben erläuterten Komplexitätsabschätzung (bzw. -relation) zu kommen.

Agile Schätzverfahren

In der agilen Softwareentwicklung haben sich Verfahren etabliert, die gerade bei der Beschätzung im Team helfen, einen schnellen Schätzvorgang zu gewährleisten. All diese Verfahren haben zum Ziel, mit möglichst wenig Aufwand eine relative Komplexitätsschätzung zu erreichen. Als Schätzskala hat sich hier die oben angesprochene abgewandelte Fibonacci-Folge durchgesetzt.

Im Folgenden beschreibe ich drei von mir ausgewählte Verfahren, die unter agilen Teams verbreitet sind und die in unseren agilen Softwareentwicklungsteams bei //SEIBERT/MEDIA ebenfalls Anwendung finden. Dabei erläutere ich jeweils, wie wir selbst diese Verfahren einsetzen. Von Unternehmen zu Unternehmen und von Team zu Team kann es durchaus Unterschiede im praktischen Einsatz geben.

Grundregeln

Alle Verfahren sehen vor, dass derjenige, der die Anforderungen stellt (in Scrum beispielsweise der PO), sowie das komplette Entwicklungsteam anwesend sind. Die Anforderungen sollten stets in Papierform (Karten) vorliegen – die Handhabung von Papier aktiviert und involviert die Teilnehmer viel stärker als eine Beamer-Präsentation.

Darüber hinaus ist es auch möglich, dem Antragssteller eine Anforderung ohne Abgabe einer Schätzung als nicht schätzbar zurückzugeben. In diesem Fall ist die Anforderung in modifizierter Form erneut vorzulegen. Schätzungen werden immer dann – und nur dann – aktualisiert, wenn sich die fachlichen Anforderungen ändern. Das sollte im Laufe des Projekts immer der Fall sein, da häufig ungenaue Initialanforderungen für die Umsetzung konkretisiert werden müssen.

Der Klassiker: Planungspoker

Jeder, der schon ein bisschen mit agiler Softwareentwicklung zu tun hat, kennt es, das „Pokern um Aufwände". Deshalb sei dieses Verfahren zuerst genannt. Die Regeln des Planungspoker sind in Kasten 2 beschrieben.

Der größte Vorteil dieses Verfahrens besteht darin, dass jedes Teammitglied seine Schätzung völlig unbeeinflusst abgibt – jedenfalls theoretisch. Allerdings zeigt die Praxis, dass es mit der unbeeinflussten Beschätzung nicht immer so weit her ist. Bei der Vorab-Befragung des PO kommt es hin und wieder zu Kommentaren der Mitspieler wie, „Puh, das ist aufwändig" oder umgekehrt „Das ist ja einfach", was einen dann natürlich in der persönlichen Beschätzung beeinflusst. Hier ist der ScrumMaster gefragt, solche Dinge möglichst zu unterbinden. Zusätzlich ist die gemeinsame Definition einer festen Timebox wichtig – sonst kann es (vor allem bei Verwendung der ausführlichen Variante des Pokerns, siehe Kasten 2) unter Umständen recht lange dauern, bis ein gemeinsames Ergebnis erreicht ist.

Einen großen Nachteil sehe ich außerdem darin, dass man hier nicht erst mit einer Reihenfolge (Sortierung) arbeitet, wie wir es natürlicherweise (ohne feste Regeln) tun würden (siehe nochmals das Beispiel in Kasten 1). Dadurch fällt es den Teilnehmern schwerer, das gerade zu schätzende Feature im Hinblick auf die Gesamtheit der Anforderungen einzusortieren.

Anforderungskartenspiel: Das Estimation Game

Das *Estimation Game* (vgl. [Röp09]) erinnert in Sachen Ablauf und Regeln tatsächlich an ein Kartenspiel wie Rommé oder Mau-Mau. Es folgt diesen Spielregeln:

- *User-Storys auf Karten mitbringen:* Der PO bringt seine Features (User-Storys) als Kartenstapel mit zur Beschätzung.
- *Reihenfolge herstellen:* Das erste Ziel besteht darin, auf dem Tisch eine Reihenfolge der Karten nach aufstei-

Abb. 2: Estimation Game: eine Reihenfolge erzeugen.

gender Komplexität zu erzeugen (siehe Abbildung 2): Ganz unten in der Folge liegt die einfachste, ganz oben die komplexeste. Das Team muss sich nun einigen, wo „oben" und wo „unten" ist.

- *Das Team spielt:* Die Teammitglieder sind nun reihum am Zug. Jeder Spieler macht einen von zwei möglichen Zügen: Der Spieler *zieht* eine Karte vom Stapel, liest diese vor und stellt dem PO Verständnisfragen. Daraufhin legt er sie an eine von ihm gewählte Stelle innerhalb der Reihenfolge ab. Oder der Spieler *verschiebt* mit einer kurzen Begründung (ein Satz) eine bereits auf dem Tisch liegende Karte an eine andere Stelle in der Reihenfolge.

- *Herumwandernde Karten:* Wandert eine Karte in der Reihenfolge hin und her, muss der PO sie aus dem Spiel nehmen. Offenbar ist die Anforderung nicht exakt genug beschrieben und es besteht Uneinigkeit im Team, welchen Inhalt das Feature genau hat.

Ist die Reihenfolge hergestellt, geht es im nächsten Schritt darum, Größenverhältnisse zu beschreiben, diese Reihenfolge also um Story-Point-Beschätzungen zu ergänzen:

- *Referenz definieren:* Das Team muss zuerst ein Referenz-Story mittlerer Größe (ca. in der Mitte der gefundenen Reihenfolge) auswählen, die es einigermaßen gut überblicken kann (Beispiel: Deutschland bei der Beschätzung der Ländergrößen).

- *Referenz beschätzen:* Für die gewählte Referenz muss nun eine Beschätzung abgegeben werden. Das kann entweder ohne spezielles Verfahren durch Konsens im Team oder auch durch eine schnelle Pokerrunde erfolgen. Sinnvolle Werte aus der Praxis liegen zwischen drei und fünf Story-Points.

- *Referenz beibehalten:* Ist das Referenz-Feature einmal gewählt und beschätzt, muss es bei folgenden Schätzmeetings stets als Referenz herhalten und sollte immer mit in die Reihenfolge (erster Schritt) einsortiert werden – und zwar unabhängig davon, ob es bereits realisiert wurde oder nicht.

- *Skala ergänzen:* Nun durchläuft man die Reihenfolge, vom Referenz-Feature ausgehend, erst nach unten (kleiner) und fragt das Team, ab wann die nächste Stufe in der Skala erreicht wird (Entscheidung im Konsens). Ebenso verfährt man anschließend in die Gegenrichtung.

Dieses Verfahren kommt in unseren Entwicklungsteams sehr oft zur Anwendung. Das Herstellen einer Komplexitätsreihenfolge aller vorgelegten Feature-Wünsche fällt den Teams erfahrungsgemäß häufig leichter, als jede Story wie beim Poker einzeln zu bewerten.

Ohne Worte schneller schätzen

Als drittes Verfahren soll die magische Beschätzung (*Magic Estimation*, vgl. [Cam10], siehe Abbildung 3) angeführt werden, die ursprünglich von Boris Gloger vorgestellt wurde. Die Spielregeln zu diesem Verfahren sind in Kasten 3 nachzulesen. Boris Gloger erklärt die Methode ausführlich in einem YouTube-Video (vgl. [Glo11]).

Der wesentliche Vorteil dieser Methode besteht darin, dass sich in kürzester Zeit eine große Menge an Anforderungen beschätzen lässt. Somit ist dieses Verfahren ideal dazu geeignet, auch für große Projekte bzw. lange Feature-Listen eine schnelle Initialschätzung mit minimalem Zeitaufwand zu erhalten. Zwar stellt sich in der Praxis heraus, dass es den Teams schwer fällt, komplett ohne zu sprechen zu agieren (wir tendieren eben dazu, uns erklären zu wollen). Haben sich Teams aber an dieses Verfahren gewöhnt, können sie selbst mit knapp gewählten Timeboxes ein gemeinsames Ergebnis finden, mit dem am Ende alle einverstanden sind.

Von der Komplexität zum Aufwand

In der Regel reicht dem Kunden eine Komplexitätsangabe, wie oben beschrieben, nicht aus – Story-Points sind zu abstrakt. Er will eine Aufwandsschätzung haben („Ich muss doch ein festes Budget beantragen").

- Die Fibonacci-Folge wird auf dem Tisch (oder auf dem Boden, wenn der Tisch nicht groß genug ist) ausgelegt (z. B. mit einem Satz Pokerkarten). Dabei sind die immer größer werdenden Abstände zwischen den Werten zu berücksichtigen (mehr Abstand zwischen den Werten 5 und 8 als zwischen 2 und 3).
- Der PO breitet seine Karten mit den Anforderungen auf dem Tisch aus.
- Das Team fängt an, die Karten innerhalb der Schätzskala zu verteilen. Dabei darf weder gesprochen, noch nonverbal per Gestik und Mimik kommuniziert werden.
- Ein Teammitglied darf jederzeit eine bereits zugeordnete Karte erneut verschieben.
- Wandert eine Karte ständig hin und her (*Nervous Nelly*), muss der PO sie aus dem Spiel nehmen, um sie nachträglich zu diskutieren.

Kasten 3: Magic-Estimation-Spielregeln.

Abb. 3: Magic Estimation: Stille Beschätzung.

Bei bestehenden Teams und laufenden Projekten, die praxiserprobt sind, kann eine Umrechnung anhand von Erfahrungswerten erfolgen. Über die *Velocity* („Wie viele Story-Points schafft mein Team durchschnittlich pro Sprint?“) lässt sich errechnen, was ein Story-Point zum aktuellen Zeitpunkt kostet. In der Scrum-Praxis bedeutet das, mindestens drei Sprints lang Erfahrungswerte zu sammeln, die man zur Umrechnung heranziehen kann. Zu beachten ist, dass der Wert eines Story-Point aus diversen Gründen durchaus während des Projektverlaufs schwanken kann. Nach jedem Sprint sollte dieser Wert deshalb erneut ermittelt werden.

Gibt es dagegen keine Werte aus der Praxis, was etwa zu Beginn eines Projekts der Fall ist, ist eine Umrechnung von Komplexität in Aufwand noch schwieriger. Eine erste grobe Schätzung kann dadurch erreicht werden, dass das Team einen so genannten *Task-Breakdown* (Definition der Realisierungsschritte mit Schätzung der jeweiligen Umsetzungszeit) nur für das vermeintlich gut bekannte Referenz-Feature (siehe *Estimation Game*) macht und dann den Aufwand für die anderen Anforderungen entsprechend über das Verhältnis umrechnet. Natürlich handelt es sich hierbei um einen instabilen, nicht belastbaren Wert – nicht mehr als eine initiale Hausnummer.

Fazit

Eine relative Komplexitätsbeschätzung ist sinnvoll. Vorab schon alle Eventualitäten und Schritte der Umsetzung zu kennen, ist sehr aufwändig bzw. meistens sogar unmöglich. Nimmt man diese Unsicherheit als gegebene Voraussetzung hin, ist eine schnellere Beschätzung über die Verhältnisse der Features untereinander möglich.

Da die Beschätzung in agilen Projekten immer eine Teamleistung sein sollte, ist es empfehlenswert, stets dieselben Schätzverfahren einzusetzen, um den nötigen Zeitaufwand zu minimieren. Ich empfehle agilen Teams, für ihre Schätzungen das *Estimation Game* auszuprobieren. Diese Methode hat eine hohe Akzeptanz, weil sie sich natürliche, alltägliche Vorgehensweisen bei der Bildung von Reihenfolgen zunutze macht. Gerade in Sprint-Planning-Meetings (bzw. beim *Backlog Grooming*), in denen ausschließlich die neuen Anforderungen im laufenden Projekt (im Schnitt weniger als zehn) zu beschätzen sind, führen geübte Teams mithilfe des *Estimation Games* in wenigen Minuten eine Story-Point-Beschätzung durch. Nur wenn größere neue Projekte anstehen und viele Storys zu beschätzen sind, setzen wir Verfahren wie *Magic Estimation* ein, um noch mehr Zeit zu sparen. ■

Links

[Cam10] D. Campey, Magic Estimation, 2010, siehe: campey.blogspot.com/2010/09/magic-estimation.html

[Glo11] B. Gloger, Magic Estimation, 2011, siehe: youtube.com/watch?v=QocEro1J65Y

[RöpO9] S. Röpstorff, Team Estimation Game, 2009, siehe: projekt-log.de/allgemein/team-estimation-game/

[Sch] S. Schmidt, Aufwand schätzen, ohne Aufwand zu haben, siehe: blog.schst.net/2010/06/planning-dice/