Dr. Thorsten Janning
(thorsten.janning@kegon.de)
ist Chefredakteur von OBJEKTspektrum und als Vorstand der KEGON AG einer der erfahrensten Berater bei der Agilisierung großer Entwicklungsorganisationen in Europa.

Scaling Agility: Wie führt man 100 agile Teams?

Seit Jahrzehnten kämpfen wir im Software-Business mit den gleichen Problemen: Wir produzieren zu viele und zu schwerwiegende Fehler, wir sind nicht effektiv genug bei der Entwicklung komplexer Softwaresysteme und nicht flexibel genug in der Anpassung existierender Systeme an sich ständig verändernde Anforderungen in sich schnell entwickelnden Märkten. Doch während in vielen Industriezweigen Lean Management nicht nur propagiert, sondern systematisch eingeführt wurde, kaut unsere Softwareindustrie lieber die unendliche Geschichte von mehr oder weniger iterativen Vorgehensmodellen und mehr oder weniger formalisierten Projektmanagement-Modellen durch. Das Ergebnis ist fast immer dasselbe: viele Regeln und viel Ordnung, aber wenig Effizienz und Kreativität. Allmählich aber erreicht die Nachricht von agilen Organisationen auch die Managementetagen: Sollte hier vielleicht der Hebel zur Verbesserung von Reaktionsgeschwindigkeit, Qualität und Effektivität auch großer Entwicklungsorganisationen liegen?
In diesem Beitrag versuche ich den Spagat zwischen einem theoretisch-methodischen Ansatz und einer schlanken, auf agilen Prinzipien fußenden Organisation und gebe Antworten auf ganz praktische Fragen, die sich bei der Einführung einer agilen Organisation stellen.

Immer mehr Unternehmen machen sich auf den Weg, die guten Erfahrungen mit agilen Praktiken in ihren Softwareentwicklungsteams zu verallgemeinern, um eine agile Gesamtorganisation zu bauen. Der Teufel steckt beim agilen Umbau einer großen Softwareentwicklungsorganisation aber nicht nur im global-galaktischen Managementansatz, der die folgenden Fragen beantworten soll:

- Wie transformiert man eine klassische in eine agile Organisation?
- Was bedeutet die Einführung einer agilen Organisation für das mittlere und obere Management?
- Wie berücksichtigt man HR-Themen, wie z. B. die Definition von Karrierepfaden und die Einbindung von Mitarbeitervertretungen, bei der Einführung agiler Organisationen?

Auch im Detail ist das richtig dosierte handwerkliche Geschick in der Umsetzung gefordert:

- Nach welchen Kriterien kann man die Teams schneiden?
- Wie können viele agile Teams ein gemeinsames Release beliefern?
- Wie muss ein agiles Portfolio-Management organisiert sein, damit es eine agile Organisation mit Aufträgen beliefern kann?

Das „Scaled Agile Framework“

In einem meiner frühen Projekte zum Bau einer agilen Organisation in Deutschland wurde nach der Installation der ersten agilen Teams bald der Ruf nach einem *Big Picture* laut: In diesem sollten die Zusammenhänge der wichtigsten Entwicklungsprozesse in einer großen agilen Organisation dargestellt werden. Wir haben viele Versuche gemacht, aber keiner war so gut gelungen wie das Big Picture des *Scaled Agile Framework (SAFe)* (vgl. [Lef13]), das Dean Leffingwell und seine Kollegen aus ihren Projekten herauskristallisiert haben (siehe Abbildung 1).

Die Erfahrung zeigt, dass dieses große Bild einen guten Überblick über die wichtigen Zusammenhänge von Entwicklung, Anforderungs-, Portfolio- und Release-Management gibt. Vor allem aber gibt es Orientierung bei der Formulierung und der Beantwortung vieler Fragen, die im Laufe eines größeren Transformationsprojekts auftauchen. Wir verwenden es deshalb im Folgenden auch als „Landkarte“ bei der Behandlung typischer Fragestellungen.

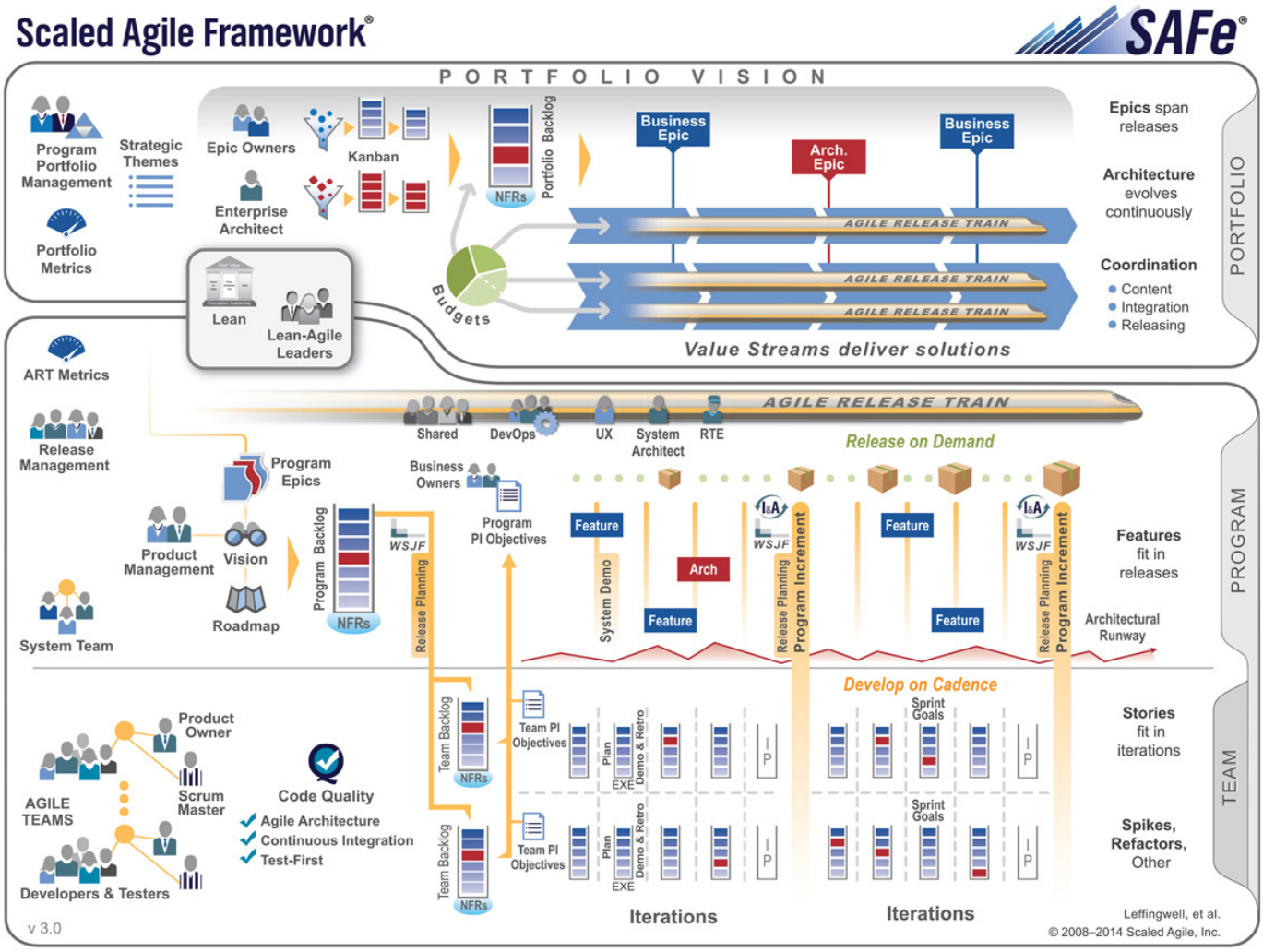


Abb. 1: Das SAFe Big Picture von Dean Leffingwell

Vom Schneiden der Teams

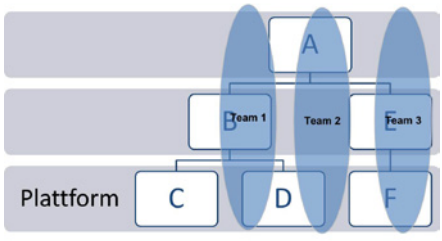
Im Kern unserer idealen agilen Entwicklungsorganisation stehen die Teams. Da wir von einer Teamgröße von circa sieben Mitgliedern (+/- drei) ausgehen, haben wir in einem Softwarehaus von circa 300 operativen Mitarbeitern eine Größenordnung von mindestens 30 Teams zu beherrschen. Im Big Picture finden wir die Teamebene ganz unten. Diese Teamebene ist übrigens die, die am besten verstanden ist, weil es inzwischen zahlreiche und umfangreiche Erfahrungen gibt.

Eine der grundlegenden Fragestellungen auf dieser Teamebene ist, nach welchen Kriterien die Teams geschnitten werden sollten (siehe Abbildung 2). Nach der agilen Lehre ist natürlich die Strukturierung nach nutzerorientierten Funktionen – also „Feature Teams“, die komplette User-Storys und nicht nur Komponenten davon realisieren können – angezeigt. Auf diese Weise wird gewährleistet, dass alle Teams am konkreten Anwendernutzen arbeiten.

Gerade in Organisationen mit einer ausgeprägten Architekturkultur steht aber schnell die These im Raum, dass wir die in Schichten strukturierte Architektur als Blaupause für den Teamschnitt nehmen, da wir auf diese Weise die Schnittstellen im Gesamtsystem optimal abbilden. Je größer das für ein spezielles IT-System benötigte Expertenwissen und die Integritätsanforderung an dieses System sind und je häufiger es von verschiedenen nutzerorientierten Funktionen, d. h. von diversen Produkt- und Service-Teams, benötigt wird, umso eher sollte dieses IT-System von einem spezifischen Komponententeam bearbeitet werden.

Es entstehen Teams für infrastrukturelle Komponenten, wie z. B. Frameworks für GUIs und Persistenz, sowie Teams, die fachliche Basiskomponenten verantworten. Real existierende komplexe Architekturen sind von teilweise multiplen Plattformebenen geprägt. Hier kann auf eine klare Verantwortung spezialisierter Teams, beispielsweise für spezialisierte Plattformen oder Komponenten, in der Regel nicht verzichtet werden, damit ein Min-

Feature-Teams



Komponententeams

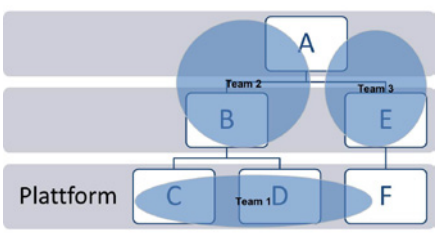


Abb. 2: Feature-Teams vs. Komponententeams

destmaß an Standardisierung gewährleistet ist. Es gibt vielfältige Beispiele für solche Architekturen: Hier sei nur auf komplexe eingebettete Software- und Hardwaresysteme verwiesen, wie sie heute z. B. in der Automobilindustrie an der Tagesordnung sind.

Ein genauerer Blick auf den Gesamtentwicklungsprozess verrät uns aber sofort die Schwäche dieses Ansatzes in Hinblick auf die Zielsetzung, flexibel und schnell auf neue Anforderungen reagieren und diese entsprechend schnell umsetzen zu können: Befinden wir uns in einem frühen Stadium der Entwicklung eines komplexen Systems, wird durch einen solchen Schnitt erfahrungsgemäß oft eine zu schwergewichtige Basisarchitektur erarbeitet, die viele Features enthält, die niemand braucht, die sich mit den ersten echten Anwendungsfällen aber als unvollständig erweist und damit zu einer komplexen und teuren Dauerbaustelle führt.

Bewegen wir uns aber in einem Umfeld mit einer komplexen existierenden Anwendungsarchitektur, die schon in Betrieb ist und die im Wesentlichen durch Erweiterungs- und Wartungsaufträge geprägt ist, wird bei einem solchen horizontalen Schnitt die Entwicklung jeder User-Story in Aufträge an zwei, drei oder mehr Teams aufgeteilt.

Wenn die Teams gleichzeitig entwickeln, entsteht folglich ein hoher Kommunikations- und/oder Koordinationsaufwand zwischen den Teams, der eigentlich durch die für die agile Entwicklung so typische direkte Kommunikation innerhalb der Teams geleistet werden sollte. Entzerren wir diesen Overhead, indem wir schichtweise nacheinander entwickeln, verzögert sich die Auslieferung der Software umso länger, je mehr wir die Entwicklung der Schichten zeitlich entzerren.

In der praktischen Umsetzung einer agilen Organisation, die in einer solchen komplexen Anwendungslandschaft arbeitet, muss man also nach Kompromissen suchen. Wenn es aus den oben genannten Gründen spezielle verantwortliche Teams für Basiskomponenten gibt, kann beispielsweise neue Funktionalität dieser Komponenten eventuell trotzdem durch die anfordernden (Feature-)Teams selbst realisiert werden. Voraussetzung dafür ist aber, dass das Know-how über den Entwurf der Basiskomponente entsprechend verteilt ist und dass die Verantwortung eines zentralen Teams trotzdem erhalten bleibt, weil sie einfache und dazu noch typisch agile Qualitätssicherungsmaßnahmen definieren: Jede Änderung eines Teams an einer Basiskomponente muss beispielsweise mittels Code-Review durch einen Spezialisten für die Basiskomponente abgenommen werden. Auf diese Weise implementieren wir eine Art teamübergreifendes Pair-Programming und bleiben damit trotz klarer Teilung der Verantwortung für Komponenten in der Architektur dem Prinzip der möglichst vollständigen Implementierung einer User-Story durch ein Team treu.

Die Erfahrung zeigt, dass dieses Prinzip nicht immer vollständig durchgehalten werden kann. Trotzdem empfehlen wir, vor dem Aufteilen von User-Storys auf zu viele Teams solche und ähnliche Varianten der flexibleren Aufgabenverteilung zwischen den Teams zu prüfen und im Zweifel anzuwenden. Jedes Aufteilen einer User-Story auf mehrere Teams erhöht noch einmal den Abstimmungsaufwand erheblich.

Planung und Steuerung der Arbeit vieler Teams

Die Notwendigkeit von Releases wird in der klassischen agilen Literatur meist nur in Bezug auf die Arbeit eines einzelnen Teams behandelt. Release-Planung bedeutet dabei im Wesentlichen, die Sprintziele auf die Entwicklungsdauer des Release einzuordnen. Genau dieses Prinzip muss auf die Koordination der Aufgaben vieler Teams verallgemeinert werden. Und so lässt sich auch ein weiteres Scrum-Werkzeug relativ einfach skalieren, auch wenn die praktische Umsetzung auf den ersten Blick vielen IT-Managern als nicht praktikabel erscheint: Wir rufen zu Beginn der Realisierungsphase für ein Release die gesamte (!) Entwicklungsmannschaft zu einer Release-Planung zusammen, auch wenn es sich um mehr als 100 Personen handelt (siehe Abbildung 3).

Nach Möglichkeit arbeiten wir zwei Tage lang daran, verschiedene Realisierungsvarianten für die Release-Ziele zu diskutieren und zu entscheiden. Außerdem erarbeiten wir technische und fachliche Abhängigkeiten und berücksichtigen diese bei der Planung der Tasks der verschiedenen Teams in den Sprints zum Release.

Zu Beginn stellen das Management und der Produktverantwortliche den strategischen Rahmen und die konkreten Ziele für das kommende Release vor. Außerdem bekommt der Chefarchitekt Gelegenheit, die wesentlichen technischen Rahmenbedingungen und anstehende Refactoring-Aufgaben zu beschreiben. Danach werden die offensichtlichen Abhängigkeiten zwischen den Aufgaben kartiert und die Teams gehen in kleinen Runden an die Arbeit, diese Abhängigkeiten im Detail zu verstehen, sie in der Abhängigkeitsmatrix abzubilden sowie die Konsequenzen für ihre eigenen Sprintplanungen festzuhalten (siehe Abbildung 4).

Nach mehreren Iterationen dieses Vorgehens und einer daraus eventuell resultie-

Abb. 3: Beispiel einer Release-Planungs-Session

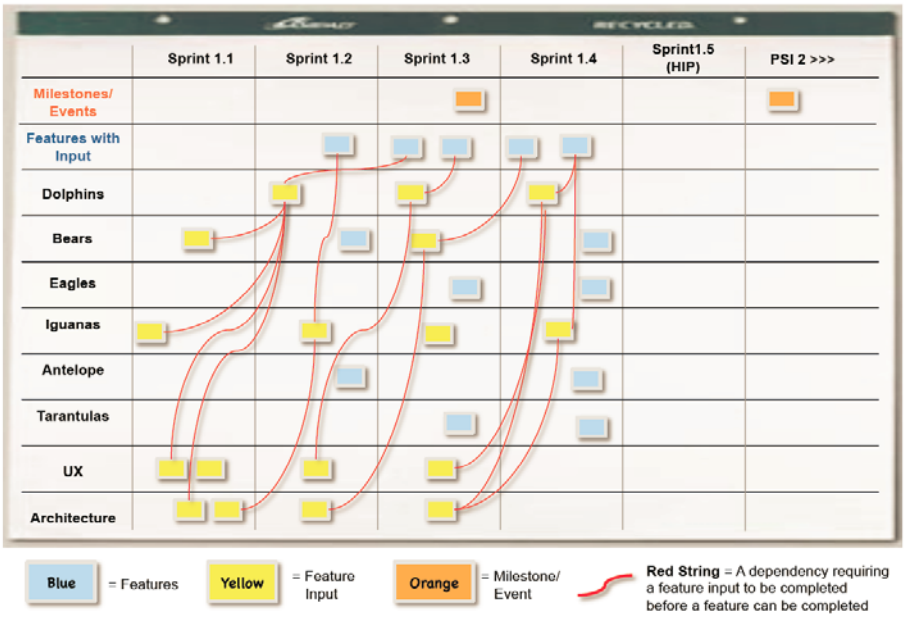


Abb. 4: Kartierung von Abhängigkeiten während der Release-Planung

renden notwendigen Korrektur des Release-Zieles durch das Management gehen die Teams wieder auseinander und starten mit ihrer Arbeit am Release. Der erste Reflex der meisten IT-Verantwortlichen auf die Vorstellung einer solchen Release-Planung ist die Befürchtung, dass der Aufwand, so viele Beteiligte zwei Tage lang planen zu lassen, die Planungskosten zu sehr ansteigen lässt und dass wertvolle Entwicklungs- und Fachkapazitäten verschwendet werden. Ein zweiter Blick zeigt aber schnell, welchen Nutzen wir durch eine so umfangreiche gemeinsame Planungssitzung haben können:

- Die Vorstellung der Vision sollte bei jedem Team normalerweise durch den *Product Owner (PO)* geschehen, damit die Teams in Kenntnis des Ziels ihre Entscheidungen in ihrer täglichen Arbeit an den richtigen Maßstäben ausrichten können. Die Tatsache, dass diese Vorstellung in einem gemeinsamen Rahmen passiert, kostet nicht mehr Arbeitszeit, gewährleistet aber ein über alle Teams hinweg gemeinsames Verständnis der Vision.
- Die Sprint-Planungssitzungen können nach einer solchen Release-Planungssitzung erfahrungsgemäß wesentlich verkürzt werden, weil der PO die Inhalte der Sprints nicht mehr so ausführlich erklären muss und man sehr schnell in einen Task-Breakdown einsteigen kann.
- Die Abhängigkeiten zwischen den Teams sind sowohl mit agilen Teams als auch in klassischen Organisationen das wesentliche Hindernis und eine der bedeutendsten Fehlerquellen. Das frühe Erkennen und Auflösen dieser Abhängigkeiten verhindert Verzögerungen und Wiederholungen in der Arbeit, die beim späteren Erkennen notwendig würden.

Man spart also entsprechend viel Arbeitsaufwand in den Sprint-Planungen wieder ein und gewinnt ein gemeinsames Verständnis und bessere Qualität. Deswegen sind die praktischen Erfahrungen mit der Durchführung solcher Planungsklausuren durchweg positiv, wenn sie entsprechend professionell vorbereitet und durchgeführt werden.

Natürlich entstehen auch nach erfolgreichen Release-Planungen unvorhergesehene Probleme, die die Planungen mehr oder weniger über den Haufen werfen. Man benötigt also eine Instanz, in der für andere Teams relevante Probleme eines Teams transparent gemacht und die Konsequenzen bzw. Risiken für die Release-Ziele diskutiert werden. Möglicherweise müssen ungeplante Features doch nachträglich eingeplant werden oder die Realisierung einzelner Features auf dem kritischen Pfad des Release-Plans stellt sich als komplexer heraus, als ursprünglich geplant. In der Regel haben die Scrum Master der Teams die Aufgabe, solche externen Abhängigkeiten ihrer Teams aufzulösen. Insofern treffen sich typischerweise die Scrum Master mit dem für das Release verantwortlichen *Release Train Engineer* in meist wöchentlichen Meetings, in denen solche Probleme behandelt werden. Wenn notwendig, werden die betroffenen Teams dann zu einer Sondersitzung zusammengerufen. Als wichtigstes Werkzeug dient dazu die Abhängigkeitsmatrix, die im Release-Planungs-Meeting entwickelt wurde.

Genauso wie bei den Sprints werden Probleme auf diese Weise früh transparent und können mit den POs diskutiert werden, sodass es keine bösen Überraschungen kurz vor der Auslieferung gibt. Entsprechend früh kann man nach fachlichen, technischen oder organisatorischen Lösungen für die auftretenden Probleme suchen. Der Lohn sind niedrigere Kosten und zufriedenere Kunden.

Portfolio-Management in agilen Organisationen

Die oberste Ebene des Big Pictures beschreibt das Portfolio-Management. Dort wird skizziert, dass man einen rollierenden Portfolio-Prozess in einem Kanban-Board organisiert. Doch wie dies praktisch zu geschehen hat, dazu gibt es unterschiedliche Interpretationen:

- Eine klassische Interpretation ist, dass das oberste Management dort seinen klassischen Portfolio-Prozess implementiert, sich aber von den festen Budget-Zeiträumen löst und Werkzeuge und Metriken, z. B. der Wertstrom-Analyse, einsetzt, um die Priorisierung der Anforderungen nachvollziehbar zu machen. Ob dann tatsächlich auch Limits für die *Work in Progress* definiert werden, ist in klassisch geprägten Organisationen häufig umstritten.
- Eine Interpretation im Sinne einer agilen Gesamtorganisation ist es, auch im Portfolio-Prozess neue Rollen, Verantwortlichkeiten und Verfahren zu implementieren. Der Teufel steckt bei dieser Aufgabe im Detail, denn in der Regel muss im Portfolio-Management nicht nur priorisiert, sondern auf dem Weg von der Idee zum IT-Auftrag auch entwickelt und konkretisiert werden.

Ein klassisches Beispiel für einen solchen Weg ist uns bei einer Bank begegnet, die beklagte, dass ihre IT zu langsam sei, um innovative Produkte schnell auf den Markt zu bringen. Ein Blick auf den Gesamtprozess zeigte aber, dass ein großer Teil der Zeit zwischen erster Produktidee und Handel mit dem Produkt auf der Ba-

sis der modifizierten IT-Systeme nicht in der IT, sondern in den fachlichen Vorüberlegungen verloren ging.

Die Implementierung eines Scrum-ähnlichen Prozesses brachte die beteiligten Fachbereiche und die IT in eine frühe intensive Kommunikation über Tasks und Abhängigkeiten zwischen Tasks. Auf dem Weg von der Idee zum Produkt konnte die Zeit der fachlichen Vorarbeiten um mehr als die Hälfte verkürzt werden. Der nächste Schritt ist es nun, die parallele fachliche Vorarbeit an mehreren Produkten auf einem Board zu organisieren.

Die Konkretisierung von der Idee zum Produkt kann dabei beispielsweise auch mit Werkzeugen des Lean StartUp (vgl. [Rie11]) erfolgen. Das hat zur Folge, dass sich das Management auf eine wirkliche Kommunikation mit dem Markt einlässt und die beteiligten Fachbereiche sich auf gemeinsame Pivot-Schritte bei der Produktdefinition einlassen. Dazu müssen die klassischen „Silo-Egoismen“ solcher Diskussionen überwunden werden.

Grenzen großer agiler Organisationen

Lassen sich auf Basis des SAFe beliebig große Organisationen „agilisieren“? Als Größeneinheit wird dort der *Agile Release Train (ART)* formuliert. Ein genaueres Hinschauen verrät, dass an einem ART maximal 100 bis 150 Entwickler arbeiten. Wir kennen diese Zahl als Dunbar-Zahl, z. B. von Mary Poppendieck (vgl. [Pop13]). Dort wird hergeleitet, dass größere Organisationen mit mehr als 100 bis 150 Menschen allein deswegen nicht so erfolgreich arbeiten können, weil dort keine effektive Kommunikation mehr möglich ist.

Wenn die Organisation größer als diese Dunbar-Zahl wird, gibt das SAFe heute keine klare Antwort, wie man eine Mehrzahl von ARTs organisieren könnte. Unsere Erfahrung sagt, dass man zwischen der oberen Ebene des Portfolio-Managements und der mittleren Ebene wieder Werkzeuge zur Koordination mehrerer ARTs implementieren kann, die ähnlich ausgeprägt sind wie die Werkzeuge zur Koordination mehrerer Teams, die in einem Release-Train arbeiten. Wie intensiv die Koordination zwischen solchen Release-Trains sein muss, hängt aber stark davon ab, wie eng die betroffenen Softwareteile verzahnt sind.

Relativ unabhängige Produkte, die vielleicht lediglich auf einer gemeinsamen Plattform basieren, können auch organisatorisch relativ lose gekoppelt werden. Damit ist es sowohl einfach als auch im Sinn einer schnellen Reaktionsfähigkeit der Organisation unproblematisch, eine weitere Abhängigkeitsmatrix zu pflegen und die Konsequenzen auf parallel arbeitende Release-Trains herunterzubrechen.

Wenn das Softwaresystem selbst aber sehr groß und eng gekoppelt ist, werden die Abhängigkeiten zwischen den Release-Trains größer – damit wachsen die Aufwände zur Koordination zwischen den ARTs. Das Risiko für ein Release gefährdende Fehler oder Anforderungen steigt damit enorm. In diesem Fall empfehlen wir eher das Grundprinzip: keine Abhängigkeiten außerhalb eines Release-Trains.

Damit wird deutlich, dass wir in erster Linie eine architektonische Aufgabe haben, die Komplexität von Softwaresystemen selbst zu begrenzen, indem wir lose gekoppelte Teilsysteme bauen. Die organisatorische Integration vieler Entwicklungsteams bleibt in ihrer praktischen Beherrschbarkeit auch mit Ansätzen wie dem SAFe begrenzt.

Im Rahmen einer Fishbowl-Diskussion mit Ken Schwaber und Dean Leffingwell über die Skalierung agiler Prinzipien berichtete ein Teilnehmer, dass er durch Ansätze wie das SAFe sehr erfolgreich die Effizienz und die Effektivität einer großen Entwicklungsorganisation verbessert hat. Dann ist er aber in eine Startup-Company gewechselt und hat gemerkt, welche enormen Effektivitätsvorteile diese kleine Organisation gegenüber der großen Organisation hatte.

Deswegen liegt es nahe, darüber nachzudenken, wie man die Arbeit architektonisch lose gekoppelter Systeme so organisieren kann, dass wirklich unabhängig voneinander arbeitende Teams einer gemeinsamen strategischen Ausrichtung folgen und beispielsweise auch gemeinsame Releases beliefern können. In [Jan09] wird die Vision einer Entwicklungsorganisation beschrieben, die zu diesem Zweck reglementierte Marktmechanismen einsetzt. Auch wenn mir noch keine real existierenden Implementierungen dieses Ansatzes bekannt sind, scheint mir dies der richtige Weg zu sein, um die Komplexität in sehr groß skalierenden Organisationen zu beherrschen. ■

Literatur & Links

[Jan09] GT. Janning, Organisation von SOA-Entwicklung und -Wartung, Euroforum-Verlag, 2009

[KEGON] Agiles Projektmanagement in großen Unternehmen, siehe: www.kegon.de/agile/

[Lef13] D. Leffingwell, Scaled Agile Framework, 2013, siehe: http://scaledagileframework.com/author/deanleffingwell/

[Pop13] M. Poppendieck, Bevor es Management gab: Warum die Dinosaurier der operativen Steuerung aussterben müssen, in: OBJEKTspektrum, 1/2013

[Rie11] E. Ries, The Lean Startup, Crown Business, 2011