



Sven Hubert

(sven.hubert@aitgmbh.de)

ist Bereichsleiter und Software Process Consultant im AIT TeamSystemPro Team und MVP für VS ALM. Er hilft Teams bei der Verbesserung der Softwareentwicklung und berät bei der Einführung und Anpassung des VS Team Foundation Server.

AGILE IM MEHRSPIELERMODUS: ERFAHRUNGSBERICHT ZU AGILEN METHODEN IM MEHRTEAM-MODUS

Sie haben Ihre Entwicklungsteams bereits auf Scrum umgestellt, die Zusammenarbeit zwischen den Teams läuft aber noch nicht reibungslos? Sie sehen sich gewachsenen Strukturen und Prozessen ausgesetzt? Ihr Produkt wird immer komplexer und weniger beherrschbar? Wie skaliert man Scrum und agile Methoden für viele Teams? In dem Artikel werden Strategien gezeigt, um gemeinsam mit den Kunden nach agilen Konzepten zu entwickeln.

Am Anfang steht der Kunde: Der hat Erwartungen, Ansprüche und ständig neue Ideen um einen Bedarf zu decken. Das ist der Grundstein für ein erfolgreiches Projekt aber auch der Beginn eines meist langen Weges. Wo genau der Weg hinführt ist meist ungewiss und nur vage bekannt. In solch einem Projekt lassen sich heute keine Entscheidungen treffen, deren endgültige Auswirkungen erst in drei Jahren bekannt werden.

Zudem treten Missverständnisse zwischen Kunden und dem Zulieferer auf. Wenn erst nach drei Jahren in ersten Tests mit dem Kunden auffällt, dass eine Anforderung falsch interpretiert und dann umgesetzt wurde, kann das teuer werden. Um dem Risiko einer Fehlentscheidung und verpassten Erwartungshaltung zu begegnen, setzen viele Unternehmen gerade in neuen Projekten auf kürzere Zeitscheiben, in denen eine vorzeigbare Teillösung der Software entsteht und dem Kunden gezeigt wird. Idealerweise ist diese Teillösung ein auslieferbares Produktinkrement, also bereits produktiv einsetzbar – das muss aber nicht sein. Schließlich kommt es darauf an, das Feedback des Kunden zum derzeitigen Stand und zur Richtung der Umsetzung zu bekommen. Nur dadurch wird erkennbar, ob weitere Ideen umzusetzen oder scheinbar bekannte Anforderungen umzudefinieren sind. Zudem können eigene Designentscheidungen verifiziert und überdacht werden. Diese Zeitscheiben werden bei Scrum als *Sprint* bezeichnet und greifen die Prinzipien des kontinuierlichen Lieferns und Reviews auf. So verwenden Kunden in Projekten beispielsweise Sprints von drei Wochen Länge. **Abbildung 1** zeigt dieses Vorgehen im Modell.

Um den Kunden nicht mit zu häufigen Zwischenlieferungen zu überfordern, ist man allerdings dazu übergegangen, lediglich alle zwei bis drei Sprints eine Version für Kundenfeedback, sprich ein *Release*, bereitzustellen. Stattdessen prüfen interne Tester die Sprint-Ergebnisse kontinuierlich bereits während der Sprints auf ihre funktionale Korrektheit, um die Qualität lange vor dem Release sicherzustellen. Dabei sind sowohl Entwickler mit automatisierten Tests als auch Tester mit manuellen oder teilautomatisierten Akzeptanztests in der Pflicht.

Erkenntnisse und Feedback zu den Produktinkrementen – sowohl aus dem internen Sprint als auch vom Kunden oder Beta-Testern – wandern zurück in das *Product Backlog* (dt. Produktrückstand). Dieser wird vom *Product Owner (PO)* gepflegt,

der in Scrum die Rolle des entscheidungsbefugten Produktmanagers einnimmt.

Soviel zu Scrum wie es allgemein gelehrt und in Abwandlungen auch häufig schon gelebt wird. Es gibt aber kaum ein Unternehmen, das nur aus einem Entwicklungsteam und immer verfügbaren POs besteht. Häufig gibt es ein ganzes Produktportfolio, viele Teams und eine Menge Leute, die mitreden wollen, wenn es um das Produkt geht. Wie skaliert man agile Methoden und Frameworks wie z. B. Scrum in einem solchen Umfeld?

Mit Strukturen brechen

Nach Conway's Law (vgl. ([Wik]) sind Organisationen dazu verdammt, Produkte zu entwerfen, die ihren inneren Strukturen aus Abteilungen und Teams folgen. Wenn

Continuous Feedback Continuous Delivery

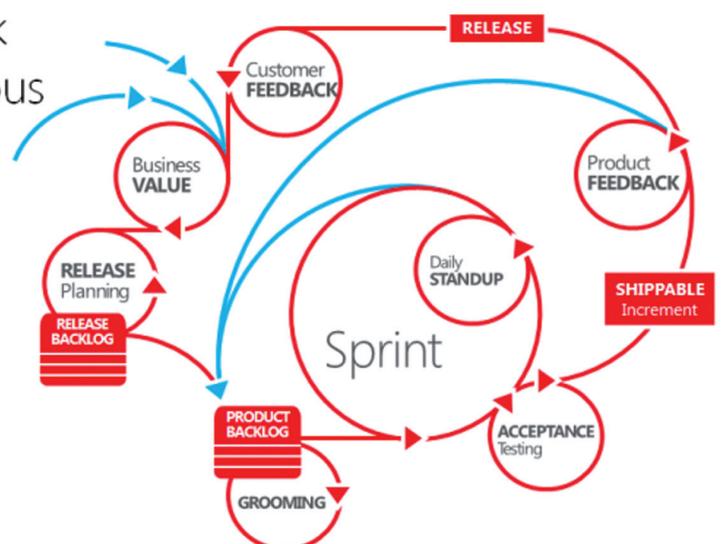


Abb. 1: Agiler Produktzyklus.

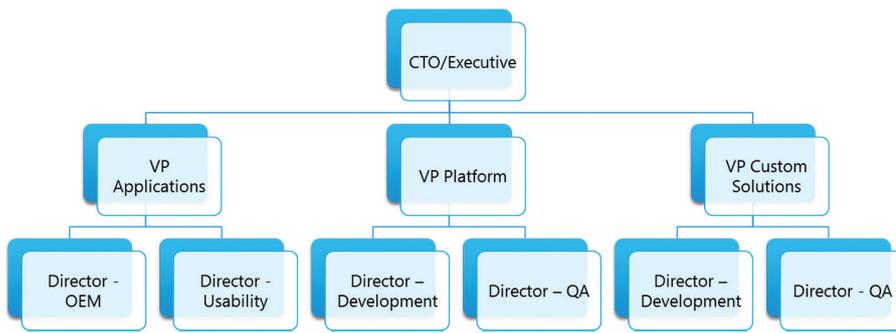


Abb. 2: Beispielorganisationsstruktur.

das Produkt schon vorhanden ist und eine neue Struktur aufgebaut wird – die Organisation beispielsweise wächst –, ist dies ebenfalls ein Thema.

Abbildung 2 zeigt exemplarisch die Organisationsstruktur eines modernen Unternehmens, wobei das Beispiel auf die Softwareentwicklung reduziert wurde. Hier wurde zwar auf oberer Ebene nicht einfach in Linien wie Entwicklung und Test gedacht, sondern zunächst in die grobe Produktstruktur unterteilt, doch manifestiert der Aufbau das Silo-Denken der Bereiche für Applikationen, Plattform und Kundenlösungen. Die Kultur denkt hierarchisch: Mitarbeiter sind Ressourcen und werden als solche minutiös in Projektplänen getaktet. Ein Mitarbeiter ist heute in diesem und morgen in einem anderen Team eingebunden – meist reagiert er auf aufgetretene Probleme und Engpässe, statt vorausschauend und geplant zu agieren. Eigentlich rein technische Entscheidungen werden in den oberen Hierarchieebenen getroffen und nach „unten“ kommuniziert. Die Teams fühlen sich nicht wirklich verantwortlich, halten Informationen zurück und reagieren häufig mit Schuldzuweisungen auf Probleme, die Terminverschiebungen verursachen. Und obwohl einige Teams bereits Scrum praktizieren, mag das Ganze nicht so recht in Fahrt kommen. Neben den genannten kulturellen Gründen liegt das aber auch an den verwobenen Strukturen des Produkts. Teams haben untereinander viele Abhängigkeiten und der Koordinationsaufwand, um ein paar Dutzend Teams zu verwalten, ist immens. Es entsteht eine Projektleitungsabteilung, die nur noch mit dieser Koordination beschäftigt ist, technisch aber nicht mithalten kann.

Zugegeben, das obige Szenario fasst viele schlechte Erfahrungen verschiedenster

Kunden zusammen, ist aber dennoch nachvollziehbar und leider realistisch. Im Folgenden sollen Wege zur Verbesserung aufgezeigt werden.

Portfolio nach Zwiebelprinzip

Eine wesentliche Verbesserung lässt sich erreichen, indem unausgesprochene Verantwortlichkeiten und Entscheidungskompetenz klar definiert und kommuniziert werden. Unser Ansatz folgt dem Modell einer Zwiebel, wie in Abbildung 3 dargestellt, wobei jede Schicht eine andere Verantwortungs-, Zeit- und Inhaltsebene darstellt.

Die Strategie wird von der Unternehmensführung festgelegt. Hierbei entstehen Ideen und Visionen zu Produkten, in die investiert wird. Die Strategie ist über einen längeren Zeitraum – in der Software häufig über zehn Jahre – ausgelegt.

Das Portfolio ist die Konkretisierung der Strategie in „Überschriften“ auf der gedachten Produktpackung. Hierbei gilt es zu definieren, welche zugesicherten, verkaufsfähigen Eigenschaften ein bestimmtes Produkt haben soll. Häufig wird auf dieser Ebene bereits in Produktgenerationen oder auch Haupt-Releases gedacht, die über einen Zeitraum von zwei bis drei Jahren entwickelt werden. Das Portfolio wird meist von einem Produktmanagement-Team (auch PO-Team) verwaltet.

Das Produkt wird in auslieferbaren Teilen geplant, die zeitlich den Release-Candidates (RC) zugeordnet werden. Ein RC ist eine potenziell auslieferbare Version des Produkts und nicht als Testversion aufzufassen. Die Verantwortung für das Produkt liegt beim PO, der aber immer noch vom PO-Team unterstützt wird.

Auf Ebene des Release werden die auslieferbaren Produktteile weiter heruntergebrochen. Hierfür können beispielsweise die in der agilen Welt beliebten User-Stories zum Einsatz kommen, wie dies auch in unserem Beispiel der Fall ist. Hier ist endgültig der PO in der Verantwortung, Prioritäten und Inhalte festzulegen und erster Ansprechpartner des Teams zu sein.

Bis jetzt liegt also die Verantwortung, die Strategie des Managements mit den richtigen Produkten und treffenden Produkteigenschaften umzusetzen, komplett beim Produktmanagement.

Auf der technischen Seite gibt es aber noch eine Entwicklungsleitung oder ein



Abb. 3: Hierarchie des Product Backlogs von der Strategie zur täglichen Arbeit.

Team von Architekten, die die Verantwortung haben, die Software langfristig adaptierbar zu halten, damit die Produkteigenschaften auch in vertretbarem Aufwand implementiert werden können.

Zudem müssen – je nach Produktgröße – *Integrationsteams* gebildet werden, die die Abhängigkeiten und kritischen Pfade zwischen verschiedenen Teams im Auge behalten und übergreifende Prüfungen und Tests durchführen können.

In den *Sprints* und damit der *täglichen Arbeit* geht die Verantwortung zum Team über, das sich aus Entwicklern, Testern, Usability-Experten und Designern zusammensetzen kann. Das Team ist für die pünktliche und vollständige Lieferung der von ihm zu Beginn eines Sprints zugesicherten User-Stories verantwortlich. Die Vorgaben vom Produktmanagement können während der Sprint-Planung als „zu groß“ klassifiziert und geändert werden. Damit ist das Team in der Pflicht, nicht zu bewältigende Aufgaben vorab zu melden und nicht kurz vor Abgabe eine Terminverschiebung einzugestehen. Doch wie strukturiert man die Teams?

Structure follows Function

Unter Berücksichtigung von Conway’s Law müssen Organisationsstrukturen den funktionalen Produktstrukturen folgen. Die Produktstrukturen müssen dazu bewusst definiert werden, um die Ziele der Organisation – wie Wachstum, Flexibilität und Langlebigkeit – zu erfüllen.

Ein großes Problem dabei sind Abhängigkeiten der einzelnen Produktmodule. **Abbildung 4** zeigt eine beispielhafte Abhängigkeitsstruktur der Module von drei Applikationen und einer gemeinsamen Plattform. Eine Auswahl von zwei Applikationen bildet das Produkt „Basic Suite“; das Produkt „Premium Suite“ ergänzt diese mit der dritten Applikation.

Im vorliegenden Fall wurden fünf Teams definiert: je ein Team für eine Applikation und zwei Plattform-Teams. Die Teams haben die Verantwortung über bestimmte Module der Software. Das widerspricht auf den ersten Blick dem *Feature-Driven-Development*, hat aber den Vorteil, dass Module sauber gehalten werden. Gibt es keinen dedizierten Verantwortlichen für ein Softwaremodul, wird es schnell verfallen, d.h. gegen Codekonventionen und Qualitätsvorgaben verstoßen und nicht mehr wartbar sein. Die Abhängigkeiten zwischen

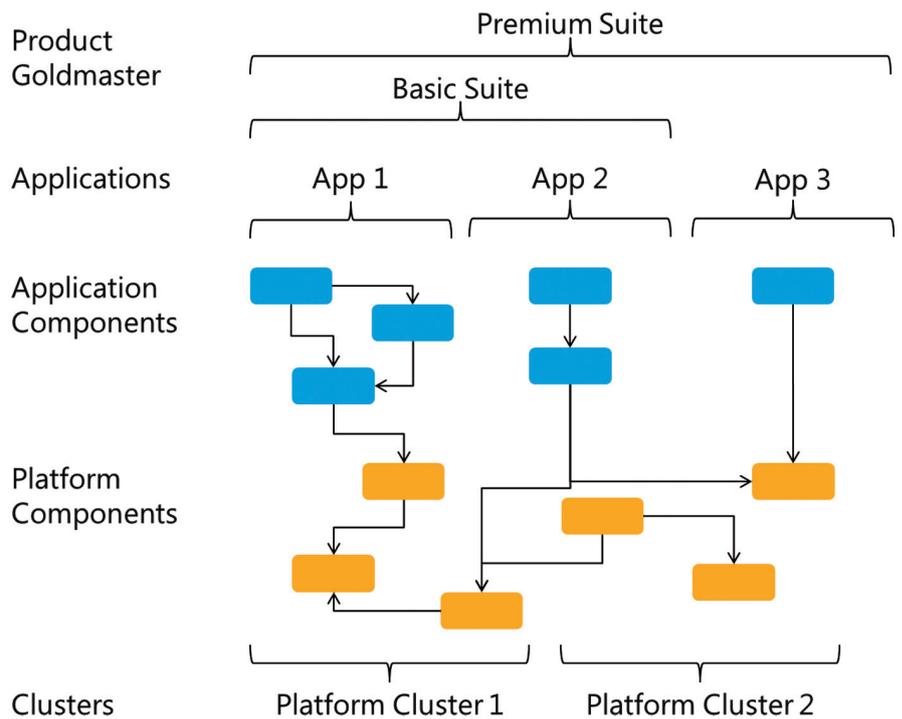


Abb. 4: Produkt- und Teamstruktur.

den Teams müssen aber minimiert werden, damit die Rechnung aufgeht. Ein Team ist entweder von einem Modul eines anderen Teams abhängig oder das andere Team konsumiert ein Modul des eigenen Teams. Es besteht nie eine zirkuläre Abhängigkeit im Sinne der Modulabhängigkeiten zwischen zwei Teams. Das kann man in **Abbildung 4** an den beiden Plattform-Teams erkennen. Das Team „Plattform Cluster 1“ ist Lieferant für „Plattform Cluster 2“.

Diese Entwicklungsteams sind multifunktional aufgebaut. Neben Entwicklern und Testern gibt es – je nach Modulverantwortung – spezifische Experten. Wird ein Oberflächenmodul gepflegt, gibt es einen Usability-Experten. Ist ein Treibermodul in der Verantwortung des Teams, hat es einen Kernel-Experten. Die Teams sind dabei durchaus unterschiedlich groß – von zwei bis zwölf Mitarbeitern.

Besonders wenn ein Team viele konsumierende Teams beliefert, ist es sinnvoll, Regeln für die Zusammenarbeit zu formulieren und schriftlich festzuhalten. Einige Unternehmen haben intern eine Art *Service-Level-Agreement (SLA)* abgeschlossen. In diesem, vielleicht vier bis sechs Seiten starken Dokument ist festgehalten, was eine *Anfrage für eine Änderung* an einem Modul enthalten muss, wer

Änderungen beschließen kann und wie diese anderen Konsumenten mitgeteilt werden. Microsoft arbeitet hier beispielsweise nach dem *Tell&Ask-Mode*. In einem Release-Team aus Architekten unterschiedlicher Teams werden vor einem bestimmten Meilenstein Änderungen an Schnittstellen lediglich angekündigt – der *Tell-Mode*. Die Konsumenten müssen darauf reagieren. Nach dem Meilenstein wird in den *Ask-Mode* gewechselt, wonach der Lieferant geplante Änderungen im Release-Team vorstellen muss. Nur wenn alle Konsumenten einverstanden sind – also noch reagieren können –, kann die Änderung noch eingebaut werden. In anderen Fällen muss nach einem Kompromiss gesucht werden.

Zudem ist im SLA geklärt, wie *Lieferungen* aussehen müssen: Lieferungen enthalten Quellcode und Binärdateien, eine Dokumentation der statischen und dynamischen Schnittstelle, Code-Beispiele für die Anwendung, ein Testprotokoll, Unit-Tests für den Konsumenten, mit denen potenzielle Fehler vor der Meldung reproduziert werden müssen usw.

Neben den modulatorientierten Entwicklungsteams gibt es (siehe oben) weitere Teams, die zum einen top-down die definierte Produktstrategie auf die Straße bringen und die zum anderen bottom-up die

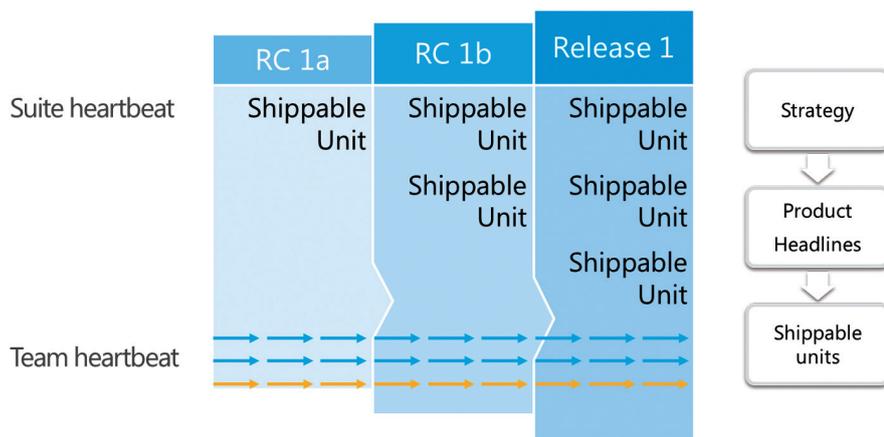


Abb. 5: Der Produktpuls.

Einhaltung von gemeinsamen Konventionen sicherstellen sowie teamübergreifende Prüfungen durchführen.

Top-down ist das Suite-Product-Management Team (PM) in der Pflicht, die Strategie in Produktfunktionen und auslieferbare Teile zu übersetzen. Es besteht aus einem Director PM und den Produktmanagern. Wichtig ist nicht unbedingt der technische Einblick – der kann von Experten erbracht werden, die in der Planung selektiv hinzugezogen werden –, sondern die gemeinsame Priorisierung und Beantwortung wichtiger Fragen wie z. B:

- Welche Kundenwünsche werden berücksichtigt, welche Funktionen werden bewusst weggelassen?
- Wohin entwickeln sich die Kunden und damit der Markt?
- Welche Funktionen müssen heute berücksichtigt werden, die der Benutzer morgen benötigt und von denen er heute eventuell noch nichts weiß?

Das Suite-Integration-Team (Int) nimmt eine technische und prüfende Rolle ein. Es installiert die Produkte regelmäßig in produktionsähnlichen Umgebungen und führt produktweite Tests durch. Das Integration Team automatisiert zudem die auszuführenden Integrationstests und arbeitet an Querschnittsthemen wie Build-Automatisierung, Codeanalyse und den Entwicklungswerkzeugen. Eine sinnvolle Ergänzung der Verantwortlichkeit des Teams ist die Prüfung von Codekonventionen und Architekturvorgaben, die von den Architekten kommen.

Bei einem unserer Kunden wurde ein „Architecture Office“ innerhalb des Inte-

gration Teams etabliert, das nach einem festgelegten Zeitplan Module auf Einhaltung der Regeln und Konventionen testet. Werden Verstöße entdeckt, werden die betreffenden Entwickler oder Teams eingeladen, um die Regel zu erklären bzw. über Gründe für den Verstoß zu diskutieren. Das hilft langfristig dabei, das Qualitätsverständnis der Entwicklungsteams auf das gleiche Niveau zu bringen.

Im Suite-Release-Team (Rel) kommen ausgewählte Vertreter sowie der jeweilige PO eines jeden Entwicklungsteams zusammen. Diese sind verantwortlich dafür, die unvermeidbaren Abhängigkeiten zwischen den Teams zu koordinieren und in einem überschaubaren Rahmen zu halten. Zudem werden technische Änderungen und Probleme besprochen (siehe Ask&Tell-Mode).

Die Vielzahl der Teams kann zu einer großen Zahl an notwendigen Meetings führen. Dem muss mit einer effizienten Meeting-Kultur und einem wohlgedachten Zeitplan begegnet werden. Das ist jedoch nicht Gegenstand dieses Artikels

(mehr dazu finden Sie beispielsweise in „Effective Daily Meetings“ (vgl. [Dai]).

Der Puls der Organisation

So viel zur Struktur. Lassen Sie uns einen Blick auf die Zeitdimension werfen. Nicht nur durch statische Abhängigkeiten zwischen Teams entstehen Koordinierungsaufwände, zeitliche Abhängigkeiten müssen ebenfalls verwaltet werden. Auch hier gilt der Ansatz der Vermeidung.

Aufwände entstehen durch häufige und unregelmäßige Übergaben zwischen Teams sowie durch den Wechsel von Mitarbeitern zwischen Teams und die dadurch aufwendigere Kapazitätsplanung. Beides sind keine unmittelbar wertschöpfenden Aktivitäten. Es hat sich herauskristallisiert, dass das Zusammenspiel deutlich einfacher wird, wenn alle im gleichen Takt denken und handeln – sozusagen im Puls der Organisation. Auch diese „Herzschläge“ lassen sich wieder auf die verschiedenen strukturellen Ebenen beziehen.

Abbildung 5 zeigt den Produkt-Suite-Puls, der über mehrere Release-Kandidaten bis hin zum Release schlägt. Das Produkt wird sukzessive um auslieferbare Teile erweitert, bis eine marktreife Version – das Release – entsteht. Die auslieferbaren Teile werden von den Entwicklungsteams im Team-Puls den Sprints geliefert. Wichtig ist, dass alle Teams ungeachtet vom Produkt in diesem Rhythmus arbeiten, um den Austausch von Mitarbeitern sowie Lieferungen zwischen den Teams zu vereinfachen. Ansonsten würde ein Mitarbeiter unter Umständen mitten in einem Sprint das Team wechseln müssen.

Das Modell ist aber flexibel genug, um den Teams einen gewissen Freiheitsgrad zu lassen. So kann es je nach Kontext des Teams sinnvoll sein, den Sprint auf die

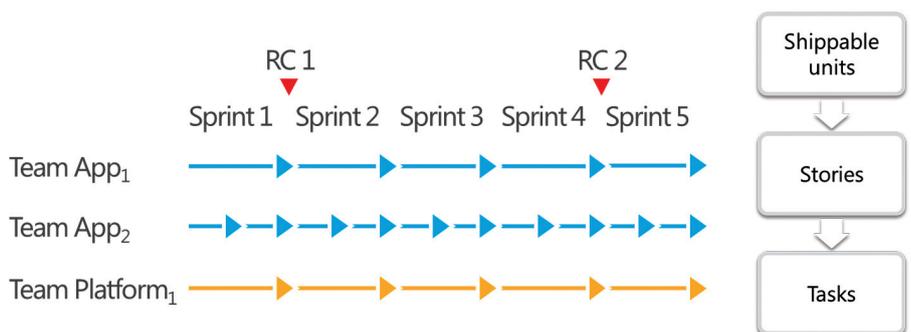


Abb. 6: Der Teampuls.

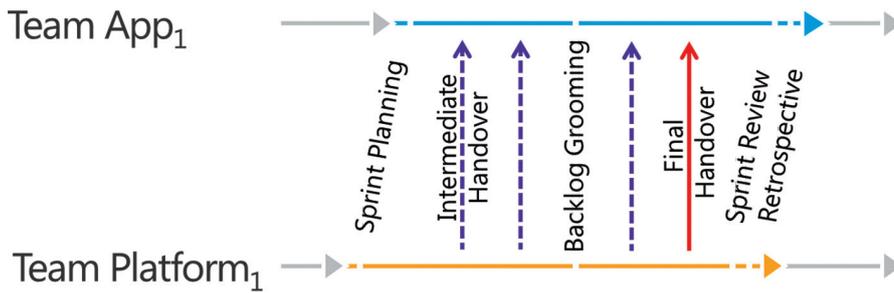


Abb. 7: Der Übergaberhythmus.

Hälfte des regulären Sprint-Zeitraumes zu verkürzen. So können gerade bei neu begonnen oder unklaren Projekten schneller Ergebnisse betrachtet und adaptiert werden. **Abbildung 6** zeigt den gleich getakteten Team-Puls und wie Team „App 2“ in Sprint 1.1 und Sprint 1.2 arbeitet. Zugleich bringen die Release-Kandidaten die Ergebnisse der Teams über das Integration-Team wieder zu einem kompletten Produkt

zusammen. So kann sich eine Routine herausbilden und damit eine kontinuierliche Effizienzverbesserung erreicht werden.

Das soll aber nicht bedeuten, dass die Integration der gesamten Produkt-Suite nur zum Zeitpunkt der Erstellung des Release-Kandidaten erfolgt. Die Entwicklungsteams integrieren regelmäßig – mindestens wöchentlich – ihre Teilergebnisse in die Integrationsumgebung des Gesamtpro-

dukts. Dort prüft und testet das Integration-Team, um frühzeitig Feedback an die Teams geben und Probleme identifizieren und beheben zu können.

Wenn man noch weiter in den Sprint abtaucht, wird die Gleichtaktung etwas aufgeweicht. **Abbildung 7** zeigt schematisch die Zusammenarbeit eines Applikationsteams „App 1“ und zuliefernden Plattfornteams „Platform 1“. Nehmen wir einmal an, dass die Applikation eine Anforderung an die Plattform stellt und bereits im Release-Team abgesegnet wurde. In der Sprint-Planung von „Platform 1“ muss diese Anfrage berücksichtigt und eingeplant werden. Dazu ist ein Vertreter von „App 1“ abgestellt, der der Sprint-Planung beiwohnt. Daher finden die Planungs-Meetings der Plattform auch ein bis drei Tage vor den Applikationen statt. Somit startet der Sprint für die Plattfornteams auch wenige Tage vor der Applikation. Entsprechend verschieben sich auch das Sprint-Ende und damit die Abgabe nach vorn.

Scaled Agile Framework™ Big Picture

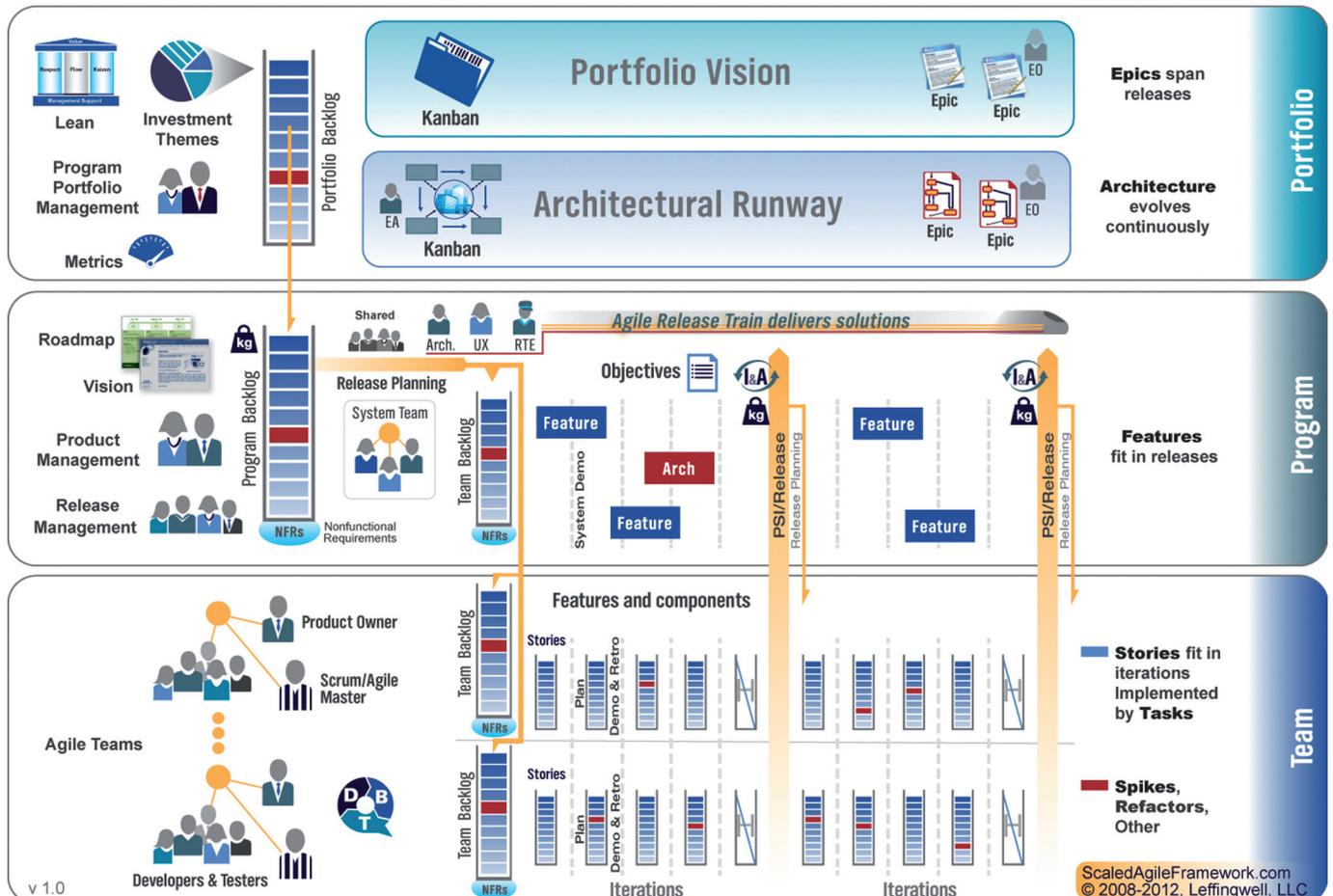


Abb. 8: Das Scaled Agile Framework (©2012 Leffingwell, LLC).

Da eine Interface-Änderung notwendig wird, bleibt dem Applikationsteam am Ende des eigenen Sprints damit noch genügend Zeit für die finale Integration der Plattformänderung (*Final Handover*). Damit das reibungslos von statten geht, werden vorher individuelle Zwischenübergaben (*Intermediate Handover*) vereinbart. Die Planung der Lieferungen bleibt aber den Teams überlassen. Auf einer höheren Ebene wird wieder in auslieferbaren Teilen und idealen Sprints gedacht. Das reduziert den Planungsaufwand in komplexen Systemen aus verschiedenen Teams und Produktteilen deutlich.

Im Takt der Sprints finden entsprechend Backlog-, Grooming- und Retrospektiven-Meetings der Teams statt. Häufig sind auch diese zeitlich so eingeplant, dass Vertreter aus konsumierenden Teams in die Meetings der Zulieferer eingeladen werden können, ohne dass diese das eigene Meeting verpassen. Auf Ebene der Release-Kandidaten erfolgen diese Meetings im Release-Team.

Die formelle Basis

Bei unserer Unterstützung von Kunden, derartige Strukturen und Prozesse zu etablieren, lehnen wir uns an das *Scaled Agile Framework* an, das Teile der oben gezeigten Konzepte abstrahiert und so eine formale Basis bietet. Es muss jedoch in der Praxis an die Bedingungen und vor allem an die etablierte Begriffswelt eines Unter-

Literatur & Links

[Dai] Daily Coaching Blog, Skip Angel, Effective Daily Meetings, siehe: igvisible.com/2011/07/effective-daily-meetings/

[Guc12] S. Guckenheimer, N. Loje, Visual Studio Team Foundation Server 2012: Adopting Agile Software Practices, From Backlog to Continuous Feedback, Microsoft Windows Development 2012

[Sca] Scaled Agile Framework, Leffingwell, LLC, siehe: scaledagileframework.com

[Wik] Wikipedia, Convey's Law, siehe: en.wikipedia.org/wiki/Conway's_law

nehmens angepasst werden, um zum Erfolg zu führen.

Das Framework in **Abbildung 8** bezieht sich dabei rein auf Softwareentwicklungen. Eine zusätzliche Herausforderung in unseren Projekten bilden häufig Unternehmen, in denen die Software nur eine untergeordnete Rolle spielt und in denen auch Hardwareentwicklungen zum Produkt beitragen. Hier wird oft argumentiert, dass die Hardwareentwicklung nicht nach Scrum läuft. Ungeachtet dessen, dass Scrum durchaus auch für die Hardwareentwicklung sinnvoll eingesetzt werden kann, ist das auch nicht wichtig. Im vorgestellten Modell ist es unwesentlich, nach welchem Vorgehen die Teams innerhalb der Takte arbeiten. Es ist sogar denkbar, dass für bestimmte Teams der Release-Kandidat die Zeitscheibe für Lieferungen darstellt, an deren Ende integriert werden muss. Jedoch birgt das die Gefahr, dass nicht passende

Lieferungen und unbedachte Probleme zu spät entdeckt werden. Das ist natürlich nicht im Sinne des Erfinders.

Fazit

Das vorgestellte Modell zur Skalierung einer agilen Vorgehensweise in großen Organisationen konzentriert sich im Wesentlichen darauf, den gestiegenen Koordinationsaufwand zu verringern. Das wird zum einen über eine der Funktion folgenden Organisationsstruktur erreicht. Zum anderen liegt der Schwerpunkt auf dem gemeinsamen Rhythmus der Teams und deren Lieferungen. Die Gleichtaktung verlangt aber auch ausreichende Freiheitsgrade, um Lieferungen zwischen Teams effizient zu gestalten. Als Basis dient das *Scaled Agile Framework*. Die Einführung eines solchen Vorgehens verlangt der Organisation allerdings einiges ab und stellt den schwierigeren Teil dar. ■