Scrum mit zwei, drei, vier … Teams

Sieben Lösungspatterns für große agile Projekte

Große agile Projekte sind nicht einfach kleine agile Projekte mit mehr Personen: Aus dieser Tatsache ergeben sich zahlreiche Fragen, wenn ein Projekt wächst: Wie skalieren Backlog und Product Owner? Wie verteilen die Teams die Arbeit? Wie tauschen Mitglieder verschiedener Teams Wissen aus und wie erkennen sie teamübergreifende Hindernisse? Dieser Artikel zeigt Lösungspatterns für sieben Themenfelder, die schnell zum Stolperstein eines großen agilen Projektes werden können.

Das klassische agile Team nimmt neue Anforderungen in Form von User-Storys auf, schätzt deren Komplexität, plant sie in einen Sprint ein und setzt sie schließlich um. So weit, so gut. Leider sind einem einzelnen Team dabei Kapazitätsgrenzen gesetzt, schließlich sollten in einem Scrum-Team nicht mehr als neun Personen zusammenarbeiten (siehe [Sch16], S. 6).

Was passiert, wenn ein Produkt umfangreicher wird oder die Projekt-Velocity für ein wichtiges Produkt erhöht werden soll? Da das Team nicht mehr Features entwickeln kann, kommen weitere Teams hinzu. Hier bewegen sich viele – mitunter gut eingespielte – Scrum-Teams auf unbekanntem Terrain: Wie werden Aufgaben über mehrere Teams verteilt? Wie synchronisieren sich Teams untereinander und wie beseitigen sie teamübergreifende Hindernisse, sogenannte Impediments? Wie wird ein einzelner Product Owner (PO) der Masse an Anforderungen und User-Storys gerecht?

Dedizierte Frameworks wie SAFe [SAFe] oder LeSS [LeSS] sind eigens für skalierte agile Projekte entwickelt worden. Für manche Teams oder Projekte sind sie jedoch zu groß; vor allem, wenn Teams nur eine schnell zu implementierende Lösung für ein spezifisches Problem suchen.

In dieser Übergangsphase arbeiten wir mit Lösungspatterns, die Antworten auf die oben angesprochenen Fragen geben können. Jedes der sieben Patterns ist ein Vorschlag: Ein Team probiert ihn aus, beobachtet die Zusammenarbeit, bewertet nach einiger Zeit, ob das Pattern wirkt, und entscheidet dann: Übernimmt es die Methode fest in den Arbeitsalltag oder nicht. Wenn die Patterns nicht oder nicht mehr ausreichen, kann ein Team ein ausgewachsenes Framework wie SAFe oder LeSS für skalierte agile Projekte einführen.

Feature-Teams setzen fachliche User-Storys um

Wenn ein zweites Scrum-Team dazu kommt, stellt sich die Frage: Wie werden Aufgaben, Storys und Verantwortlichkeiten in Zukunft verteilt? Zufällig? Schnell bilden sich Spezialisierungen heraus, oder die Teams entscheiden sich für einen Aufgabenschnitt nach Komponenten. Derartige Komponenten-Teams betreuen eine oder mehrere Komponenten des Systems, etwa die Präsentationsschicht, die Business-Logik oder die Datenbank, und entwickeln auf ihrem Teilgebiet spezielle Expertise. – Das hört sich gut an.

In der Praxis bedeutet die Organisation in Komponenten-Teams allerdings einen technischen, horizontalen Schnitt durch das Gesamtsystem. Das zieht oft ein Backlog aus technischen Storys nach sich, die meist nur einen Teil eines kundenzentrierten Features umsetzen. Das widerspricht der Kundenorientierung, die Scrum einführt. Ein weiterer Nachteil: Features mit Berührungspunkten zu mehreren Komponenten werden von mehreren Teams bearbeitet. Das kann zu unerwünschten Abhängigkeiten zwischen den Teams führen.

Im Gegenentwurf sind Feature-Teams für je ein Feature verantwortlich (**siehe Abbildung 1**). Features durchdringen mehrere oder alle Schichten des Systems. Jedes Team bearbeitet so ein breites Feld an Themen. Statt spezieller Expertise braucht es einen *generalizing-specialist*-Ansatz.

Die Organisation in Feature-Teams ist ein fachlicher, vertikaler Schnitt durch das System. Er baut auf einem Backlog aus echten, fachlichen User-Storys auf, die den Geschäftswert in den Fokus rücken; das löst ein wichtiges agiles Prinzip ein.

Die Teammitglieder eines Feature-Teams bauen breites Wissen auf allen Ebenen des Technologie-Stacks auf. Teams werden flexibler und weniger abhängig voneinander. Auch die Bearbeitungszeiten für Features verkürzen sich in der Regel, wenn Teams nicht auf Ergebnisse eines anderen Teams warten. Außerdem fördern Feature-Teams die gemeinsame „Ownership" des Codes: Jeder im Team fühlt sich für den Code insgesamt verantwortlich.

Im Gegenzug brauchen die Mitglieder von Feature-Teams mehr Wissen und Fähigkeiten als in Komponenten-Teams: Sie verstehen idealerweise die Fachlichkeit, um fachliche User-Storys umsetzen zu können, und sie decken als ganzes Team alle Komponenten ab, statt sich auf wenige Schichten zu spezialisieren. Die Teammitglieder brauchen Zeit, um Wissen weiterzugeben und aufzunehmen. Ihnen helfen agile Techniken wie Pair- oder Mob-Programming [Zui14]; sie fördern den schnellen Wissenstransfer.

Der Koordinationsaufwand zwischen Teams verschiebt sich: Komponenten-Teams arbeiten häufig gemeinsam an einer Funktion und sprechen dafür konti-

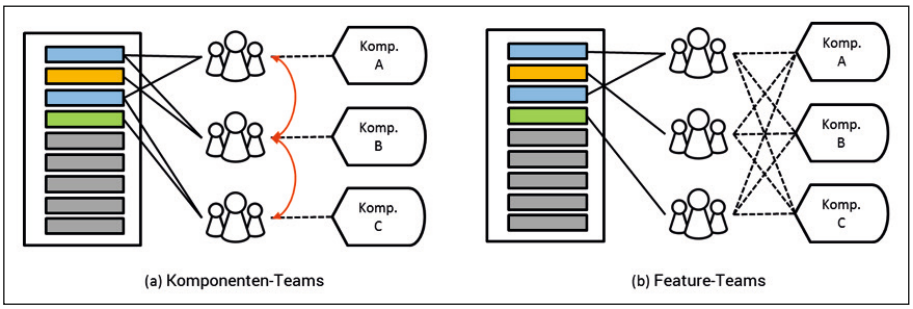

Abb. 1: In der linken Grafik koordinieren sich drei Komponenten-Teams auf Team-Ebene, in der rechten Grafik arbeitet jedes der drei Feature-Teams auf Quellcode-Ebene

nuierlich teamübergreifend miteinander. Der Abstimmungsbedarf liegt auf der interpersonellen Ebene und dreht sich zum Beispiel um die Übergabe von Projekten. Zwischen Feature-Teams liegt der Abstimmungsaufwand stärker auf der technischen Ebene: Jedes Team bearbeitet sein isoliertes Feature. Es muss allerdings in den gemeinsamen Quellcode integriert und mit den Features der anderen Teams zusammengeführt werden. Continuous-Integration-Praktiken wie Versionsverwaltung, Test- sowie Build-Automatisierung bieten hier jedoch etablierte Lösungen.

Ein großes agiles Projekt sollte nach Möglichkeit von Anfang an Feature-Teams etablieren. Existiert bereits ein Schnitt nach Komponenten, werden sie sukzessive in Feature-Teams transformiert: Zunächst wird ein einzelnes Feature-Team aus Vertretern mehrerer Komponenten-Teams gebildet, um die nötigen Kenntnisse im Team zu vereinen. Glückt dieses Experiment, werden Schritt für Schritt weitere Feature-Teams aufgebaut.

Communities of Practice arbeiten an der gemeinsamen Architektur

Agilität lebt von technischer Exzellenz und einer guten, konsistenten Architektur, die Adaptionen zulässt und Wartbarkeit gewährleistet. In einem großen agilen Projekt ist das eine besondere Herausforderung, da mehrere Teams unabhängig voneinander an der Architektur arbeiten und sie erweitern – insbesondere mit Feature-Teams. Spezielle Architektur- oder Plattform-Teams sichern eine einheitliche Architektur oder verantworten die Nutzung eines Frameworks. Sie widersprechen jedoch dem kundenzentrierten Ansatz von Feature-Teams und bauen neue Abhängigkeiten zwischen Teams auf.

Eine Alternative ist eine Community of Practice (CoP): eine themenspezifische Gruppe, die sich quer zu den Teams etabliert. Architekturinteressierte Kollegen tauschen sich hier regelmäßig über die Architektur der gemeinsamen Anwendung aus. Mit Communities of Practice können Teams auch andere übergreifende Anliegen angehen, etwa Testing, Infrastruktur oder Usability. Jeder Vertreter trägt das gemeinsam erarbeitete Wissen in sein jeweiliges Team zurück und ist Ansprechpartner für sein Thema. Die Verantwortung für die Entwicklung eines Features und der dazugehörigen Architektur liegt dann wieder autonom im Team.

Startet ein Projekt mit einem einzelnen Scrum-Team, bietet sich zudem das Konzept des Tiger-Teams an. Das erste Team entwickelt zunächst Software und Kernarchitektur. Überschreitet die Arbeitslast die Kapazitäten des initialen Teams, wird das Tiger-Team zugunsten mehrerer Feature-Teams aufgelöst. Die Mitglieder des Tiger-Teams teilen sich auf, sodass in jedem Feature-Team mindestens ein ehemaliger „Tiger“ ist. Er gibt die Grundzüge und Feinheiten der entwickelten Architektur an die Projekt-Neulinge weiter.

Ein zentrales Backlog

Im Produkt-Backlog werden alle Anforderungen an ein Produkt organisiert und verwaltet. Der Product Owner pflegt es kontinuierlich und priorisiert Storys. Im Planning übernimmt das Team in Absprache mit dem PO die User-Storys mit der höchsten Priorität in das Sprint-Backlog des kommenden Sprints. Arbeiten mehrere Teams an einem Produkt, entstehen schnell mehrere Backlogs – jedes Team plant dann mit einem eigenen. Solche Team-Backlogs laufen inhaltlich jedoch schnell auseinander und verschleiern die Sicht auf das Gesamtprodukt.

Um den Vorteil eines zentralen Produkt-Backlogs zu erhalten, lohnt sich die Unterteilung in Produkt- und Sprint-Backlog (**siehe Abbildung 2**):

- Das *Produkt-Backlog* enthält alle Anforderungen und User-Storys an das Produkt in der Reihenfolge, die der PO vorgibt, ohne Zuordnung zu einem konkreten Team.
- Im Planning zu Beginn des Sprints übernimmt jedes Team seine Storys in das eigene *Sprint-Backlog*. So hat jedes Team während des Sprints nur die eigenen Aufgaben im Blick.

Dem übergreifenden Produkt-Backlog mit der Gesamtheit an Epics und dazugehörigen User-Storys stehen in jedem Sprint mehrere Sprint-Backlogs als temporäre Arbeitssicht für die Teams gegenüber. Im Planning bleibt so die Gesamtsicht erhalten, im Sprint können die Teams durch die gefilterte Sicht effektiv arbeiten.

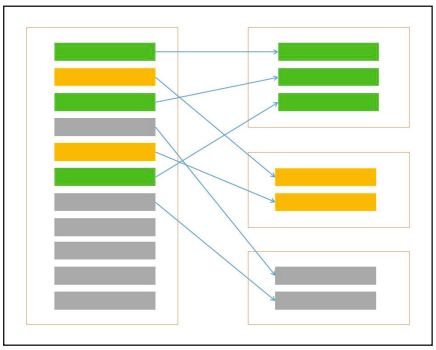

Abb. 2: Ein Gesamtbacklog wird in mehrere Sprint-Backlogs aufgeteilt

Delegation von Aufgaben in das Team

Der PO verantwortet in Scrum das Produkt-Backlog, er entwickelt und schneidet User-Storys, pflegt das priorisierte Produkt-Backlog und nimmt umgesetzte Storys ab. Mit mehreren Teams steigt jedoch seine Arbeitsbelastung, er wird vermutlich nicht mehr alle Aufgaben selbst erfüllen können. Der erste Impuls ist, jedem Feature-Team einen PO zuzuweisen. Damit verteilt man jedoch die zentrale Entscheidungsgewalt des PO auf eine Gruppe von Personen – die sich einigen müssen, welche Features sie priorisieren. Bei einem zentralen Produkt-Backlog kann das schnell zu Konflikten führen.

Alternativ lassen sich Aufgaben in das Team delegieren. Das entlastet den PO, und er bleibt zentrale Entscheidungsinstanz. Statt ausgearbeiteter User-Storys legt der PO beispielsweise nur eine kurze, informelle Beschreibung der Anforderungen vor. Ein Teammitglied formuliert sie in eine User-Story mit dazugehörigen Akzeptanzkriterien aus. Das Teammitglied benutzt dazu die in Scrum für Arbeiten außerhalb der Entwicklung vorgesehene Zeit und klärt Rückfragen direkt mit dem Kunden. Der PO prüft die spezifizierte User-Story nur noch – spätestens im Planning, besser früher.

Wird das Projekt beziehungsweise Produkt auch dafür zu groß, verantworten Area-POs jeweils einen in sich zusammengehörigen Bereich des Produkts. Der Bereich kann dabei durchaus mehrere Epics umfassen. Sie schreiben eigenständig User-Storys, pflegen sie ins Backlog ein und nehmen realisierte Storys für ihr Feature ab. Über den Area-POs gibt es einen einzelnen Chef-PO, der nur noch auf Epic-Ebene arbeitet und in Konfliktsituationen als Entscheidungsinstanz fungiert.

Reviews auf dem Marktplatz

Scrum sieht zum Ende jedes Sprints ein Review vor, in dem der PO die umgesetzten User-Storys abnimmt. Interessierte Stakeholder können teilnehmen und offene Fragen klären. Das Review skaliert allerdings nur bedingt, da es bei mehreren Teams länger dauert und die anwesenden Stakeholder und Teammitglieder länger bindet – obwohl sie möglicherweise nicht an allen Punkten interessiert sind.

In einem Bazar-Review suchen Stakeholder demgegenüber Austausch mit den

Teams, deren Inkrement sie interessiert. Jedes Team zeigt Interessierten an einem Stand die im Sprint abgeschlossenen Storys beziehungsweise den eigenen Anteil am Inkrement des abgelaufenen Sprints. Der PO nimmt die Storys der Teams während des Bazar-Reviews Stand für Stand ab. Interessierte Stakeholder und Teammitglieder können sich währenddessen ebenfalls von Stand zu Stand bewegen und Fragen stellen.
So kann der PO in kurzer Zeit viele User-Storys von unterschiedlichen Teams abnehmen. Andere Beteiligte informieren sich durch den Bazar-Stil interessenbezogen. Das Review behält so seine zentrale Rolle für alle Beteiligten, ohne sie zu langweilen.

Scrum of Scrums verteilt Wissen

Kommunikation und Wissenstransfer sind elementare Prinzipien von Agilität. Im Daily Scrum tauschen sich Mitglieder eines Scrum-Teams täglich für 15 Minuten effizient über Fortschritte, Ziele und Hindernisse jedes Einzelnen aus. Probleme von oder Abhängigkeiten zwischen Teammitgliedern können so schnell identifiziert und gemeinsam oder vom Scrum Master beseitigt werden.
In einem agilen Projekt mit mehr als einem Team braucht man zusätzlich den Austausch über Teamgrenzen hinweg. So identifizieren die Beteiligten Abhängigkeiten zwischen Teams oder führen neue Komponenten ein. Wenn der informelle Austausch nicht effektiv ist, schafft das sogenannte Scrum of Scrums einen dedizierten Rahmen für teamübergreifenden Wissenstransfer: Zwei- bis dreimal pro Woche nimmt ein Abgeordneter pro Scrum-Team an dem zusätzlichen Regelmeeting teil. Er kommuniziert analog zum Daily Scrum Fortschritte, Ziele und Hindernisse seines Teams, die Teilnehmer diskutieren aktuelle Probleme im Projekt. Vertreter für das eigene Team ist immer diejenige Person, die potenzielle Probleme im Projekt zum aktuellen Zeitpunkt am besten versteht und kommentieren kann. Je nach Art des Problems ist das entweder der Scrum Master, ein Entwickler oder ein Tester. Ideal ist es, wenn die Teilnehmer rotieren: Es nimmt immer derjenige teil, der sich im Sprint intensiv mit einem Thema auseinandergesetzt hat.
Keinesfalls sollte Scrum of Scrums zu einem Statusreport-Meeting für das Management oder zu einem Koordinierungstermin der Scrum Master degenerieren. Ziel ist die Inspektion und Adaption der aktuellen Arbeit durch die Teams im Hinblick auf die Erreichung des Sprint-Ziels. Ergebnisse – etwa Schnittstellen zwischen Teams oder die Regelung von Verantwortlichkeiten – können dafür in einem zentralen Impediment-Board gepflegt werden.

Gemeinsame Retrospektiven gegen Hindernisse

Agile Softwareentwicklung adaptiert und verbessert Prozesse kontinuierlich, um Produktivität, Effizienz und Qualität zu steigern. Regelmäßige Retrospektiven sind in Scrum das Mittel der Wahl: Das Team trägt Erfahrungen aus dem abgelaufenen Sprint zusammen, reflektiert positive wie negative Aspekte der Zusammenarbeit und leitet konkrete Maßnahmen ab, um den Prozess mit Blick auf den kommenden Sprint zu optimieren. In großen agilen Projekten braucht es ähnlich wie beim Review eine Möglichkeit, teamübergreifende Prozesse zu bewerten.
Drei verschiedene Varianten von Retrospektiven haben sich als hilfreich erwiesen:

- In einer *Joint Retrospective* am Ende jedes Sprints diskutieren der PO, die Scrum Master sowie ein Vertreter aus jedem Team über die teamübergreifende Zusammenarbeit, gemeinsames Lernen der Teams oder teamübergreifende Strukturen wie die Communities of Practice.
- Eine andere Möglichkeit ist die *Retrospective of Retrospectives*. Sie legt den Fokus auf die in den Team-Retrospektiven festgelegten Maßnahmen. Ziel ist, diese zu ergänzen, zu verbessern, zu re-priorisieren oder durch andere Teams zu unterstützen.
- An einer *Projekt-Retrospektive* schließlich nehmen alle Teams in kompletter Besetzung teil. Mit Techniken wie Story-Telling und Open-Space wird der Projektverlauf reflektiert, anschließend werden gemeinsame Themen und Probleme analysiert. Erkenntnisse und Verbesserungsvorschläge werden konsolidiert, priorisiert und in Maßnahmen übersetzt.

Hindernisse, die in gemeinsamen Retrospektiven identifiziert werden, können in einem zentralen Impediment-Board transparent gemacht werden. Da Impediments in der Regel unvorhergesehen auftreten, eignet sich ein klassisches Kanban-Board besser als ein Scrum-Board. Der Fokus bei der Impediment-Beseitigung liegt auf der schnellen Lösung der Probleme – ein Ziel, das ein Kanban-Board unterstützt.

Vom Pattern zum Framework

Ein agiles Team, dessen Projekt langsam größer wird und das mehr Personen benötigt, wird ohne aktives Gegensteuern mit einem oder mehreren der angedeuteten Skalierungsprobleme konfrontiert. Die hier vorgestellten Patterns sind nicht der einzige Weg zu einem funktionierenden skalierten Projekt. Es sind potenzielle Bausteine, die die Reibungsverluste verringern können. Getreu dem agilen Manifest und seinem Grundwert „Individuals and interactions over processes and tools“ handelt es sich jedoch nicht um Lösungen, die in jedem Kontext funktionieren und deswegen mechanisch und isoliert vom Team- sowie Projektkontext angewendet werden.

Wenn in einem Team eines oder mehrere der beschriebenen Probleme auftauchen, lohnt es sich, eines der beschriebenen Patterns für einen festgelegten Zeitraum auszuprobieren – als Experiment. Nach dem definierten Ende des Experiments wird das Ergebnis bewertet: Hat sich die Veränderung des Prozesses gelohnt? Sind die vorher beklagten Probleme weniger geworden oder verschwunden? Falls diese Fragen mit „Ja“ beantwortet werden, lohnt sich die Weiterführung des Pattern. Bei einem „Nein“ wird das Pattern überdacht oder ganz eingestellt.

Reichen die entwickelten Patterns nicht aus, lohnt sich ein Blick in die beiden Frameworks für große agile Projekte. Sie führen jeweils zusätzliche Maßnahmen, Artefakte und Rollen ein. Je nach Unternehmensumgebung, in der ein Framework eingeführt werden soll, haben beide Vor- und Nachteile.

SAFe eignet sich vor allem in Unternehmen, die aus dem klassischen Umfeld kommen und bisher Projekte mit dem Wasserfallmodell durchgeführt haben. Da SAFe neben der Team-Ebene auch die Programm- und die Portfolio-Ebene ausführlich betrachtet, können bestehende Strukturen einfacher in das SAFe-Framework transformiert werden. So können sich die Mitarbeiter ohne Angst, ihre Rollen oder Aufgaben zu verlieren, auf die agilen Methoden einlassen. Das Risiko einer SAFe-Einführung besteht darin, dass die verschiedenen Rollen als Möglichkeit genutzt werden können, eine grundlegende Organisationstransformation zu vermeiden: Die Transformation hin zu agilen Methoden muss zwingend auch die Organisationsform betrachten, sonst ist die Gefahr des „alten Weins in neuen Schläuchen“ sehr groß, die Transformation droht schnell wieder an Fahrt zu verlieren.

LeSS eignet sich besonders, wenn ein neues Produkt in nicht bereits etablierten Organisationen entwickelt werden soll. Durch die direkte Anlehnung an Scrum eignet sich das Framework zudem für wachsende Projekte: Ein Team beginnt mit dem Standard-Scrum-Prozess und skaliert ihn bei Bedarf. So kann – und sollte! – das Team den Standard-Prozess durchführen und im Rahmen des „Inspect and Adapt“ nur die zielführenden Maßnahmen von LeSS übernehmen.

Egal ob Pattern oder Framework: Es lohnt sich, neue Lösungsstrategien inkrementell einzuführen, um so die Effekte einer Veränderung besser bewerten zu können. So kann das Wachstum eines Projektes Schritt für Schritt agil erfolgen. Die Modifizierungen finden nur dann statt, wenn sie einen echten Mehrwert liefern, das stellt den Menschen und die Interaktion in den Vordergrund. ||

Literatur & Links

[LeSS] B. Vodde, C. Larman, LeSS – Large Scale Scrum, 2016, siehe: **https://less.works**

[SAFe] Scaled Agile Framework, Scaled Agile Inc., 2016, siehe: **http://scaledagileframework.com**

[Sch16] K. Schwaber, J. Sutherland, The Scrum Guide, 2016, siehe: **https://www.scrumalliance.org/why-scrum/scrum-guide**

[Zui14] W. Zuill, Mob Programming – A Whole Team Approach, in: Agile2014 Conference Experience Reports, siehe: **https://www.agilealliance.org/wp-content/uploads/2015/12/Experience-Report.2014.Zuill_.pdf**

Die Autoren

Christoph Niemann
(christoph.niemann@maibornwolff.de)
ist IT-Consultant bei MaibornWolff.
Er arbeitet am liebsten im agilen Requirements Engineering an der Schnittstelle zwischen Fachbereich und Entwicklungsteam.

Timo Becker
(timo.becker@maibornwolff.de)
ist Software-Ingenieur bei MaibornWolff.
Er entwickelt Individual-Software in einem großen agilen Kundenprojekt.