Flexible Konsistenz mit der Gilde

Architekturarbeit in agilen Großprojekten

Juri Urbainczyk

Agile Projekte versprechen Flexibilität und Änderbarkeit durch inkrementelle Problemlösung in kleinen, selbstständig agierenden Teams. Gleichzeitig arbeiten Großprojekte mit mehreren heterogenen Scrum-Teams, deren Ergebnisse im Sinne langfristiger Projektziele integrierbar sein müssen. Lokale Flexibilität und langfristige Konsistenz: Wie sind diese widerstreitenden Forderungen erfüllbar? Je komplexer Projekte werden, desto mehr entpuppt sich diese Frage als Projektrisiko. Zwei Dinge haben hier entscheidenden Einfluss: eine robuste Zielarchitektur und eine Team-übergreifende Architekturinstanz mit Entscheidungskompetenz. Der Artikel beschreibt, wie beide Aspekte mithilfe einer agilen „Architekturgilde“ erfüllt werden.

Agil unterwegs

Scrum wurde zunächst für kleine Teams mit ca. sieben Mitarbeitern geschaffen und damit für Problemstellungen begrenzter Komplexität, die mit dieser Anzahl von Personen handhabbar sind. Jedoch gibt es auch deutlich größere Projekte, die unter Nutzung von Frameworks wie SAFE [SAF] oder Nexus [NEX] auf drei oder mehr Teams abgebildet werden.

Scrum-Teams setzen Anforderungen und Ideen in ihrem jeweils eigenen Kontext und Horizont um. In jedem größeren Projekt gibt es jedoch Themen, die mehrere oder alle Teams betreffen und daher koordiniert werden müssen. Sie wirken primär von außen auf die Teams und sind daher Team-intern nur schwierig herzuleiten und zu steuern. Viele dieser übergreifenden Themen (z. B. „Wie laufen Monitoring-Events durch das System?“) betreffen die Architektur beziehungsweise werden durch sie bestimmt (s. Kasten „Ar-

Das Projekt

Die hier besprochenen Erfahrungen sammelt der Autor als verantwortlicher Chefarchitekt in einem Projekt für selbstbediente Vertriebssysteme. Vier agile Scrum-Teams und insgesamt mehr als 50 Mitarbeiter arbeiten seit Ende 2016 in 14-tägigen Sprints daran, den wichtigen Vertriebskanal mit einer zukunftsfähigen Architektur und innovativen Features auszustatten. Weitere Ziele sind Modularisierung, Betriebsexzellenz sowie Kostenreduktion. Java und JavaScript sind die wesentlichen Programmiersprachen in einem leichtgewichtigen Webtechnologie-Stack. Die grafische Benutzungsschnittstelle basiert auf Angular, die Backends auf einer REST-Microservice-Architektur. Entwicklung, Test und Produktion erfolgen in der Cloud.

Juri Urbainczyk arbeitete nach dem Physik-Studium in Münster mehrere Jahre als Softwareentwickler mit C++ und Java. Seit 2009 ist Juri Urbainczyk Bereichsleiter bei der Agon Solutions GmbH (ein Unternehmen der S&N Invent) und als Architekt und Projektleiter unter anderem für BMW, Deutsche Bahn und Lufthansa tätig. Schwerpunkte seiner Arbeit sind Portal- und Web-Architekturen sowie Beratungsprojekte im Umfeld der Digitalisierung. Juri Urbainczyk ist Gründer der Community Enterprise Architektur Rhein-Main. Seine Freizeit verbringt er mit der Familie in der Wetterau oder produziert elektronische Musik.
E-Mail: juri.urbainczyk@agon-solutions.de

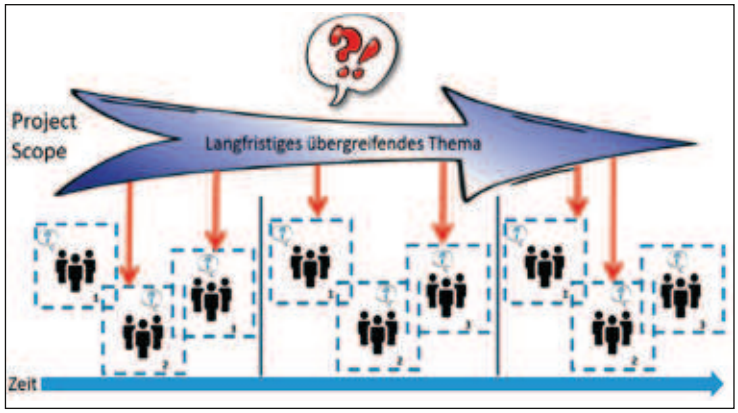

Abb. 2: Wie werden übergreifende Themen langfristig adressiert?

chitektur“). Hatte aber ein klassisches Projekt (z. B. im V-Modell) vorab mehrere Monate Zeit, solche Architekturfragen detailliert zu klären, so werden nun von Tag 1 an Lösungen von der Architektur gefordert.

Hinzu kommt, dass Scrum zwar die Rollen Product Owner (PO), Scrum Master und Entwickler kennt – aber keinen expliziten Architekten. Die Verantwortung für Architekturthemen ist vakant oder oszilliert wild durch das Team, was vor allem problematisch ist, wenn es darum geht, langfristige Konzepte einheitlich umzusetzen (s. Abb. 2). Wie mit diesen Herausforderungen umzugehen ist, dazu gibt Scrum wenig Hilfestellung.

Mehr als Scrum

Nach Erfahrung des Autors ist es nicht allein damit getan, übergreifende Themen in das Backlog zu übernehmen. Das liegt zum einen daran, dass die betreffenden Themen meist nicht rein fachlich-funktional geprägt sind und daher als Backlog-Einträge „unhandlich“ sind. Sie lassen sich z. B. häufig nicht separat schätzen, sondern immer nur in Kombination mit einer fachlichen User-Story – aber leider gibt es da nur selten ein 1:1-Mapping.

Zum anderen besitzen die Mitglieder eines Scrum-Teams nur eine unvollständige Menge an Informationen zum Gesamtprojekt und damit eine eingeschränkte Sichtweise auf die für ihr Team relevanten Themen. Daher ist eine zielgerichtete, übergreifende Steuerung alleine aus den Teams heraus problematisch.

Es besteht somit die Gefahr, dass übergreifende Themen gar nicht oder zu spät berücksichtigt werden. Zusätzlich ist die Verantwortung für Architektur nicht klar genug verankert. In der Folge geht wertvolle Zeit verloren und das Projekt handelt sich komplexe Refactorings (und mehr) ein. Muss man also über Scrum hinausgehen?

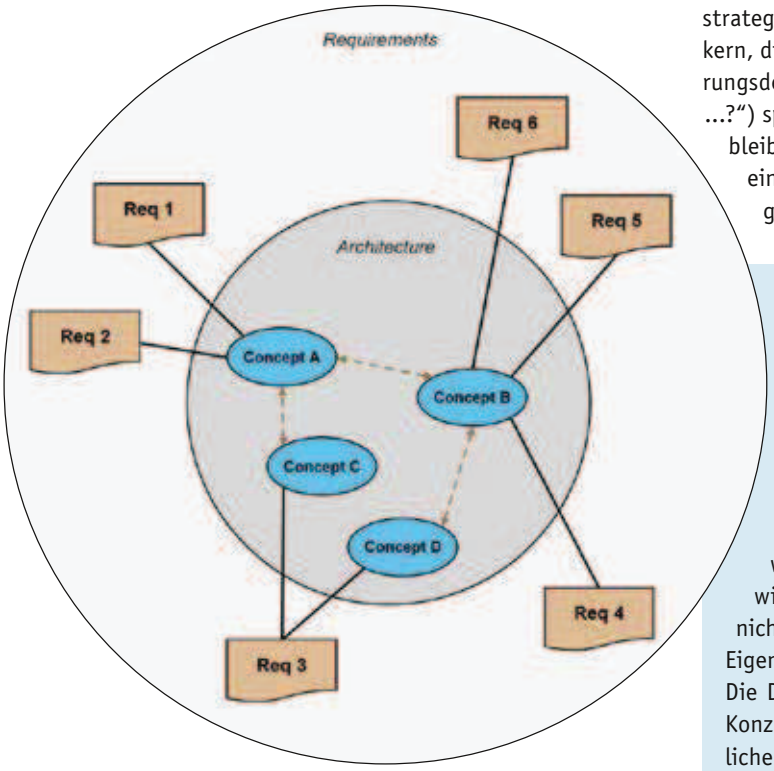

Abb. 1: Architektur als eine Menge von Konzepten

Die Zielarchitektur gibt den Rahmen vor

Viele Projektziele (z. B. „Wir brauchen durchgängige Mandantenfähigkeit!“) besitzen für alle Teams Relevanz und bleiben über einen längeren Zeitraum stabil. Daher ist es sinnvoll, passende Lösungsstrategien in einer langfristig wirksamen Zielarchitektur zu verankern, die für das gesamte Projekt Gültigkeit besitzt. Implementierungsdetails (z. B. „Welches Framework wollen wir einsetzen, um ...?“) spielen an dieser Stelle noch keine Rolle. Die Zielarchitektur bleibt bewusst auf einem höheren Abstraktionsniveau und gibt einen Rahmen vor, der den Teams Orientierung gibt und zugleich Entscheidungsspielraum lässt.

Architektur

Mit Architektur ist hier „Softwarearchitektur“ gemeint, also die Frage, aus welchen Komponenten das Softwaresystem auf welche Weise aufgebaut ist. Architektur in diesem Sinne ist eine Menge von Konzepten (s. Abb. 1), wie „UI als separate Schicht“. Die Konzepte basieren auf Architekturentscheidungen (z. B. „Die UI wird mit Angular implementiert“). Die Entscheidungen wiederum beziehen sich auf Anforderungen [JSA] – ist das nicht der Fall, wird ein goldener Henkel produziert, also eine Eigenschaft des Systems, für das es keine Anforderung gibt. Die Dokumentation „ist“ nicht die Architektur (das sind die Konzepte), sondern spiegelt sie nur in mehreren unterschiedlichen Sichten und Notationen wider [ARC].

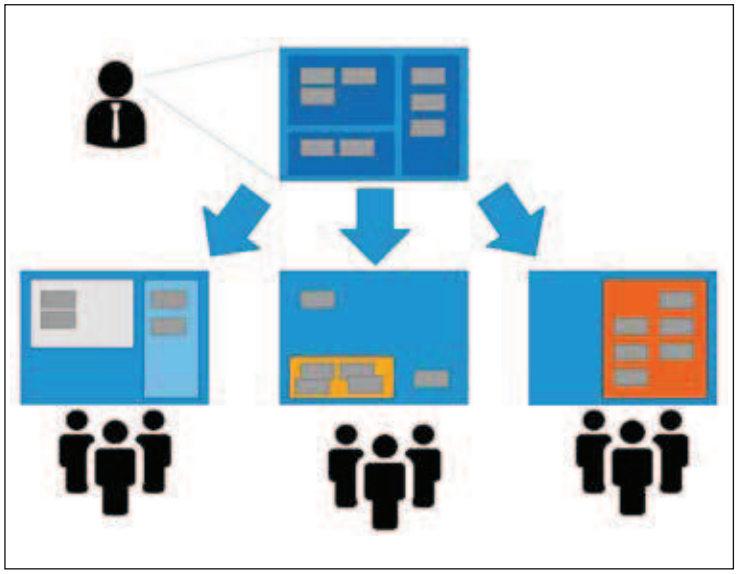

Abb. 3: Eine Vogelperspektive ist nötig, um Kompatibilität und Konsistenz sicherzustellen

Ein gutes Beispiel für eine Lösungsstrategie im Projekt des Autors (s. Kasten „Das Projekt") ist die „Trennung von Plattform und Anwendung". Damit soll primär das Projektziel einer kürzeren Time-to-Market verwirklicht werden, da die vertriebliche Geschäftslogik in der „Anwendung" von der betrieblichen und Hardware-abhängigen Logik in der „Plattform" getrennt wird. Auf diese Weise können beide Aspekte unabhängig voneinander entwickelt und gewartet werden, ohne Seiteneffekte und andere negative Abhängigkeiten zu erzeugen. Die Trennung der Plattform von der Anwendung wird in der Zielarchitektur verankert, um langfristig Wirkung zu zeigen.

Die Zielarchitektur erfüllt einen weiteren wichtigen Zweck: Sie gibt eine Struktur vor, nach der sich die Teams organisieren können. Zeigt die Zielarchitektur zum Beispiel drei grobe Cluster von Komponenten, die nur wenige gegenseitige Schnittstellen aufweisen, bieten sich drei getrennte Scrum-Teams an, die jeweils für einen der Cluster zuständig sind. Das reduziert die Abhängigkeiten zwischen den Teams und somit auch Komplexität und Risiko im Projekt.

Mikro- und Makro-Architektur

Agilität braucht vor allem Freiraum. Das heißt, Architektur-Diskussionen müssen (auch) in den Scrum-Teams stattfinden. Es geht nicht darum, die Architekturkompetenz aus den Teams herauszulösen, denn sie werden dort dringend benötigt. Jedes Scrum-Team muss in der Lage sein, die anstehenden und häufig feingranularen Entscheidungen selbst zu treffen. Ein völlig unabgestimmter Entscheidungsprozess wäre aber kontraproduktiv, denn er würde die spätere Integrierbarkeit des Ergebnisses erschweren.

Hier hilft die Unterscheidung zwischen „Mikro-Architektur" und „Makro-Architektur" [OTT]. Entscheidungen zur Mikro-Architektur (z. B. „Wie ist eine Komponenten-interne Schnittstelle aufgebaut?") trifft das Team. Entscheidungen zur Makro-Architektur, die meist von übergreifender Bedeutung sind (z. B. „Wie sieht die externe Schnittstelle der Komponente aus?"), können entweder von Projektzielen oder von den Teams als Inputgeber getrieben werden. Die Makro-Architektur ist also mit der Zielarchitektur (s. o.) gleichzusetzen.

Der Übergang von Mikro- zu Makro-Architektur ist fließend. Es gibt Grenzfälle, bei denen eine zwischen den Teams vermittelnde Instanz nötig ist, die dafür sorgt, dass keine Gegensätze oder Parallelwelten entstehen. Eine solche Instanz muss in der Lage sein, eine übergreifende Vogelperspektive (s. Abb. 3) einzunehmen und aus dieser Sicht Architekturthemen ganzheitlich zu verfolgen. Diese Aufgaben können durch eine „Architekturgilde" ausgefüllt werden.

Die Architekturgilde trägt Verantwortung

Um Architekturverantwortung in jedem Scrum-Team explizit zu verankern, gehört im Projekt des Autors zu jedem Team ein expliziter „Team-Architekt". Jeder Team-Architekt ist Teil der „Architekturgilde", die als übergreifende Architekturinstanz fungiert (s. Abb. 4). Zusätzlich gibt es einen Team-übergreifend agierenden Architekten, welcher die Gilde koordiniert und als Schnittstelle in Richtung PO und Scrum Master dient.

Der Begriff „Gilde" (s. Kasten „Gilde (historisch)") wurde im agilen Umfeld vor allem durch Spotify [SPO] popularisiert und bezeichnet dort Mitarbeiter mit gemeinsamen Interessen. Ähnlich ist es im SAFE-Framework, das von einer „Community of Practice" spricht und den Vorteil des gegenseitigen Lernens betont [COP]. Im Projekt des Autors ist eine Gilde deutlich mehr: Sie hat klar definierte Aufgaben und Verantwortungen mit Entscheidungskompetenz. Dennoch besteht sie weiterhin aus Mitarbeitern der einzelnen Teams, was die Zusammenarbeit mit den Teams vereinfacht.

Die Gilde verantwortet wesentliche Architekturaspekte; dazu gehören die Dokumentation [ARC] und Weiterentwicklung der Zielarchitektur, die Abstimmung wichtiger Projekt-externer Schnittstellen sowie das Management der Architekturrisiken. Beim Austausch mit technischen Stakeholdern (z. B. Architekten anderer Projekte) übernimmt die Gilde in Delegation einen Teil der PO-Rolle.

Eine weitere Aufgabe der Architekturgilde ist es, die Einhaltung und Umsetzung der nicht-funktionalen Anforderungen (NFA) sicherzustellen. Die NFA müssen bereits frühzeitig in den Entwicklungsprozess eingebracht werden, um sie den jeweils betroffenen User-Storys zuzuordnen. Das ist eine Aufgabe im Backlog Refinement, bei dem mindestens immer ein Vertreter der Architekturgilde anwesend ist. So kann basierend auf der besser spezifizierten Story eine sinnvolle Schätzung durchgeführt werden, die dann auch die NFA-relevanten Tasks umfasst.

Die Gilde verfolgt und steuert zudem die Abarbeitung übergreifender Themen wie Logging und Resilienz. Dazu schreibt sie eigene Epics, schätzt entsprechende Aktivitäten für die nächsten Sprints und fordert notwendige Mitarbeit von den Scrum-Teams (z. B. für Workshops oder Reviews) an. Das entsprechende Arbeitspensum

Gilde (historisch)

Eine Gilde war im Mittelalter ein Zusammenschluss von Kaufleuten oder Handwerkern zum Schutz und zur Förderung gemeinsamer Interessen. Die ersten Gilden wurden im 8. Jahrhundert nachgewiesen und versprachen ihren Mitgliedern Schutz und Hilfe in allen Lebensbereichen. Dazu zählten die Sicherheit des Warentransports und die gegenseitige Unterstützung in Unglücksfällen. Mit der Zeit gewannen die Gilden zunehmend an politischem Einfluss. Eine der bedeutendsten Fernhandelsgilden in Europa im Mittelalter war die Hanse, die sich schließlich Mitte des 14. Jahrhunderts zu einem mächtigen Städtebund weiterentwickelte.

kann dann im Planning der Teams berücksichtigt werden. Dieser Prozess ist wichtig, um Transparenz über die Kapazitäten der Mitarbeiter und gleichzeitig über den Fortschritt bei den übergreifenden Themen zu erhalten.

Ist Emergenz auch Effizienz?

Häufig wird bei agilem Vorgehen die sogenannte „Emergent Architecture“ [EME] beschworen. Diese Denkschule hofft, dass durch je nach Anlass getroffene Ad-hoc-Entscheidungen eine sinnvolle Gesamtarchitektur entsteht. Wie das menschliche Auge im Laufe der Evolution ohne Plan durch spontane Auslese-Entscheidungen entstanden ist, erwarten die Vertreter der Emergenztheorie das Gleiche von der Architektur.

Jene Apologeten des Chaos sollten sich aber klar machen, dass sogar das menschliche Auge eine suboptimale Lösung [EYE] darstellt, die sehr anfällig ist, häufig eine Sehhilfe benötigt und nur mit enormem Aufwand in einem langen Zeitraum entstanden ist. Es besteht das Risiko, dass analog auch in einem IT-Projekt hohe Aufwände entstehen und Lösungen hinterher trotzdem nicht zusammenpassen. Genau um diese Effekte abzufedern, sind Zielarchitektur und Architekturgilde so wichtig.

Kommunikation als Aufgabe

Eine zentrale Aufgabe der Gilde ist, die Kommunikation zwischen den Teams über Architekturthemen zu unterstützen und so die technische Integration der jeweiligen Ergebnisse sicherzustellen. Bei den Treffen der Gilde schaffen Berichte der Teamarchitekten gegenseitiges Verständnis und helfen, Abhängigkeiten und Unklarheiten frühzeitig zu entdecken.

Wichtig für die Wirksamkeit der Zielarchitektur ist, dass sie nicht nur im „Inner Circle“ der Gilde verbleibt, sondern in die Köpfe aller Projektmitarbeiter vordringt. Denn jede Architektur kann nur so gut sein, wie sie verstanden wird. Insbesondere die Lösungsstrategie, die Grundlage jeder Architektur ist, muss jedem Projektmitarbeiter bekannt sein.

Reicht es dann nicht, ein Wiki zur Verfügung zu stellen oder ab und an einen Newsletter zu versenden? Sind nicht alle selbstorganisiert? In der Realität haben die meisten Mitarbeiter weder Zeit

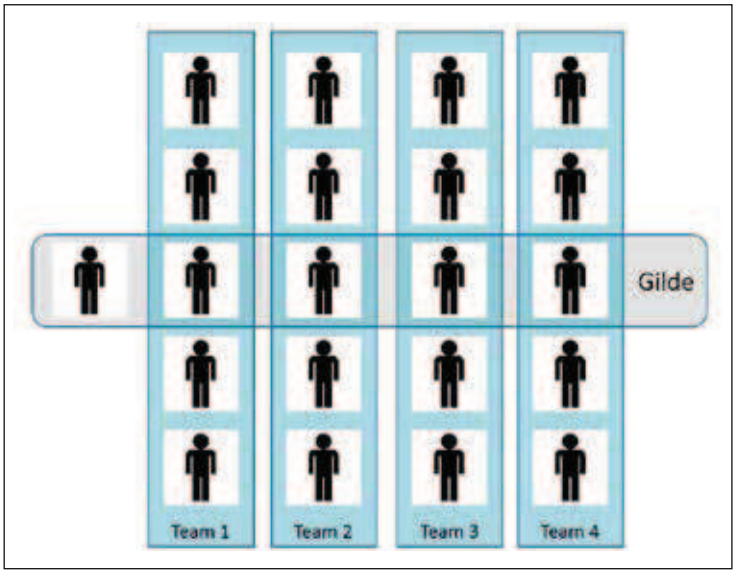

Abb. 4: Organisation der Architekturgilde horizontal zu den Scrum-Teams

noch Lust, alle notwendigen Informationen in Eigenregie zusammenzutragen. Und selbst wenn: Entscheidet dann jeder spontan, was relevant ist? Hier ist auch Effizienz ein wichtiger Gesichtspunkt: Die Gilde muss die Architektur in die Teams tragen, sei es durch Teilnahme am Planning, durch Workshops zu Spezialthemen oder durch Informationsveranstaltungen. Dahinter steht die Einsicht, dass jeder im Projekt dessen Ziele und die relevanten Architekturentscheidungen kennen muss, um zielgerichtet und effektiv arbeiten zu können.

Die Zielarchitektur agil weiterentwickeln

Die Zielarchitektur (Makro-Architektur) ist – gerade in einem agilen Projekt – nicht statisch, sondern sie entwickelt sich ständig weiter. Anfangs gibt es nur grobe Komponenten, welche dann im Projektverlauf verfeinert und ausgestaltet werden, wenn das für diese Entscheidungen notwendige Wissen erarbeitet wurde. Dieser Prozess wird vor allem durch die Scrum-Teams getrieben. Manchmal wollen verschiedene Teams unterschiedliche Änderun-

gen an der Zielarchitektur vornehmen (z. B. möchte ein Team eine NoSQL-Datenbank einführen, während andere dort eine Graphen-Datenbank sehen).

Im Falle sich widersprechender Teamentscheidungen oder falls diese im Konflikt mit der Lösungsstrategie stehen, muss die Gilde eine Entscheidung herbeiführen. Im Zweifel entscheidet sie im Sinne der gesamthaften Ziele (im obigen Beispiel wurde für die NoSQL-Datenbank entschieden, da alle Teams sie nutzen können und das Know-how im Projekt vorhanden ist). Dies ist für die Wertigkeit der Architektur ein ganz wesentlicher Prozess, denn eine „gute" Architektur zeichnet sich durch konzeptionelle Integrität aus, das heißt, sie sollte für eine bestimmte Problemstellung jeweils nur eine Lösung bereitstellen.

Leiten ohne einzuengen

Konzeptionelle Integrität heißt auch, dass alle Scrum-Teams bei ähnlichen Fragestellungen gleiche Lösungen verwenden, oder zumindest keine, die sich grundsätzlich widersprechen. Um das zu erreichen, sind technische und methodische Vorgaben (z. B. „Alle Services müssen Cloud-ready sein.") sinnvoll. Sie sind als „Leitplanken" gedacht, welche die Entscheidungen der Teams in die richtigen Bahnen lenken ohne zu sehr einzuengen. Es muss genügend Freiraum bleiben, um agil Entscheidungen in der Mikro-Architektur zu treffen und somit flexibel zu bleiben. Die Leitplanke „Alle Schnittstellen müssen automatisiert testbar sein." schreibt zum Beispiel nicht vor, wann und wie diese Tests durchzuführen sind.

Die Leitplanken können unter anderem in der Definition of Done (DoD) oder auch in Form von modifizierten User-Storys Eingang in die Arbeit der Teams finden. Leitplanken müssen akzeptiert sein und ihre Begründung sollte von jedem verstanden werden. Entsprechende Entscheidungen benötigen daher eine fundierte Abstimmung und eine Berücksichtigung aller wesentlichen Argumente. Diese Architekturarbeit geht über den Scope einzelner Teams hinaus und ist somit eine Moderations- und Konsolidierungsaufgabe für die Architekturgilde.

Die wichtigste Leitplanke im Projekt des Autors lautet „Jede Komponente in der Zielarchitektur muss auch eine Entsprechung im Quellcode besitzen." Das klingt zunächst banal, ist es aber keineswegs. In vielen Reviews und Audits hat der Autor erlebt, dass das Architekturbild und der Quellcode oft nur wenig miteinander zu tun haben. Genau das soll durch die Leitplanke zur „Architektur-Code-Entsprechung" verhindert werden. Jede Box im Architekturdiagramm muss einen Gegenpart im Code besitzen, sei es eine Klasse, ein Package, ein Interface oder Ähnliches. Falls nicht, so ist die Frage durchaus berechtigt, ob die „Box" überhaupt sinnvoll ist und nicht besser aus dem Diagramm entfernt werden sollte.

Schuldenfreie Architektur

Jedes Entwicklungsprojekt produziert im Laufe der Zeit technische Schulden, also falsch eingesetzte Entwurfsmuster, toten Code, mehrere Methoden mit gleicher Funktionalität (nur leicht anders) und so weiter. Ohne Gegenmaßnahmen bremst die Schuldenlast bald die Weiterentwicklung. In agilen Projekten führt die Ausrichtung auf kurzfristige Zyklen dazu, dass Dinge „eben mal schnell" und ohne ausreichende Abwägung erledigt werden. Das Ergebnis kann der DoD genügen – manchmal aber nur pro forma. Naturgemäß sind PO eher fachlich getrieben und nehmen die Problematik daher erst mit Verzögerung wahr. Den Scrum Mastern fehlt oft der nötige technische Background, um zielgerichtet eingreifen zu können.

Deshalb ist Continuous Integration (CI) mit entsprechend automatisierten Tests und großer Abdeckung für agile Teams unverzichtbar. Das hilft auch im Projekt des Autors, die technischen Schulden zu monitoren (z. B. mit SonarQube [SQB]). Allerdings fallen vor allem architekturrelevanten Schulden durch das Raster, denn sie sind nur sehr schwer automatisiert (z. B. durch statische Codeanalyse) zu erkennen. Hier muss die Architekturgilde entsprechende Maßnahmen wie Peer-Reviews oder Abhängigkeitsanalysen entwerfen, sodass die Architekturschulden sichtbar werden.

Dazu ein Beispiel: Eine Leitplanke empfiehlt, zwischen bestimmten Komponenten serviceorientierte Schnittstellen einzusetzen. Die Teams deuten das durchgängig als REST-Service und setzen die Schnittstellen entsprechend um. Tatsächlich wären aber bei einigen Komponenten auch Java-Interfaces ausreichend gewesen wären. Die technischen Schulden in diesem Beispiel bestehen in einem Übermaß an Entkopplung (siehe „Premature optimization is the root of all evil" [PRE]), was die Komplexität erhöht und auch die Performanz gefährden kann. Es ist unmöglich, eine solche Fehlinterpretation automatisiert zu entdecken; daher sind regelmäßige Code-Reviews notwendig. Auch Team-Externe müssen sie ab und an durchführen, da ein anderer Team-Interner sicherlich derselben Fehlinterpretation aufsäße und daher das Problem gar nicht erkennen könnte.

Fazit

Architektur trägt wesentlich zur Änderungsfähigkeit von Software bei, was in agilen Projekten die Grundvoraussetzung für inkrementelle Entwicklung ist. Eine robuste Zielarchitektur sorgt zudem für Integrationsfähigkeit und Konsistenz des Gesamtergebnisses. Die Herausforderung liegt darin, einen an langfristigen Zielen ausgerichteten Architekturprozess zu leben, ohne Flexibilität und Dynamik der agilen Entwicklung zu behindern.

Dieser Artikel hat gezeigt, wie durch eine Architekturgilde eine übergreifende Architektursteuerung möglich wird. Erfolgsfaktoren sind vor allem der moderierende Charakter sowie gleichzeitig die Verankerung der Gilde in den Teams durch die Teamarchitekten. Dabei sind Konflikte mit den PO nicht ausgeschlossen, übernimmt die Gilde doch bewusst Teile ihres Portfolios. Die Praxis zeigt, dass die Vorteile überwiegen und Scrum auf diese Weise sehr effektiv ergänzt werden kann.

Links

[ARC] http://www.arc42.de/
[COP] http://www.scaledagileframework.com/communities-of-practice/
[EME] https://intellyx.com/2015/09/28/what-is-emergent-about-emergent-architecture/
[EYE] http://www.wissenschaft.de/home/-/journal_content/56/12054/55885/
[JSA] http://juriswritings.blogspot.de/2012/03/what-is-software-architecture.html
[NEX] https://www.scrum.org/resources/nexus-guide
[OTT] https://dev.otto.de/2013/04/14/architekturprinzipien-2/
[PRE] http://wiki.c2.com/?PrematureOptimization
[SAF] http://www.scaledagileframework.com/
[SPO] http://www.full-stackagile.com/2016/02/14/team-organisation-squads-chapters-tribes-and-guilds/
[SQB] https://www.sonarqube.org