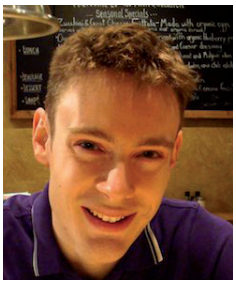
Mathias Scheffe
(M.Scheffe@insiders-technologies.de)
ist Product Owner bei Insiders Technologies und ist verantwortlich für die Entwicklung eines Business-Intelligence-Tools für den Document-Entry-Point. Seine Arbeit ist die eines Business-Analysten. Er spezifiziert zusammen mit dem Entwicklungsteam die Anforderungen an das System und vertritt dabei die Interessen der Endkunden. Im Unternehmen ist er verantwortlich für den wirtschaftlichen Erfolg des Produktes.



Balthasar Weitzel
(Balthasar.Weitzel@iese.fraunhofer.de)
ist wissenschaftlicher Mitarbeiter am Fraunhofer Institut für Experimentelles Software-Engineering und beschäftigt sich in Projekten mit Industrie- und Forschungspartnern mit der Architektur von Informationssystemen. Dabei liegt sein Fokus auf der Anwendung von wissenschaftlichem Vorgehen in der industriellen Praxis.

Erosion von Agilität mit Epic-Architekturen erfolgreich bekämpfen

Es hat sich gezeigt, dass ein agiler Ansatz, der nur durch nachträgliche Refactorings neue Features in ein Produkt integriert, nicht dazu geeignet ist, langfristig Agilität im Projekt zu bewahren. Dabei gerät das Team in einen ständigen Zyklus, bei dem viele Teile des Produktes in jedem Sprint immer wieder angepasst werden. Mit einem überlegten Einsatz von Softwarearchitektur in der agilen Entwicklung lassen sich wertvolle Vorteile erzielen, die sich sowohl lang- als auch kurzfristig auswirken. Unsere Erfahrungen, die in industrieller Produktentwicklung gesammelt wurden, bestätigen dies eindrucksvoll. Innerhalb einer Kooperation von erfahrenen Praktikern aus der Industrie und Fraunhofer-Forschern wurde gezeigt, wie solch ein Einsatz von Architektur gestaltet sein kann und welche Effekte damit zu erzielen sind. Hierbei wurde durch selektive Architekturarbeiten sowohl die Produktivität des Teams als auch der Entscheidungsspielraum des Product Owners erhöht.

Erosion von Agilität als wiederkehrendes Muster

In vielen agilen industriellen Softwareentwicklungsprojekten, die über einen längeren Zeitraum laufen, kann eine Erosion von Agilität festgestellt werden. Die Zeichen dafür sind unterschiedlich, häufig jedoch dauert die Realisierung neuer Features im Verlaufe des Projektes immer länger, obwohl deren fachliche Komplexität nicht zunimmt. Eine weitere Beobachtung ist die Zunahme der Kosten für Änderungen an der bereits realisierten Software. In direktem Zusammenhang damit steht eine Vergrößerung der Anzahl der bei einem Feature betroffenen System-Artefakte (siehe Abb. 1).

Eine eingehende Analyse der Gründe dafür führte zu der Erkenntnis, dass solch eine Erosion von Agilität mit einer Erosion von Architektur einhergeht. Um den Auslöser für diesen Verlust einer klaren Systemarchitektur zu verstehen, lohnt es sich den gesamten Lebenszyklus der Software zu betrachten.

Dieser beginnt meistens mit einem grob umrissenen Kernproblem, das die Software lösen soll. Dafür wird ein tragfähiges Konzept skizziert und initial umgesetzt. Basierend auf den Erfahrungen, die dabei gemacht werden, wird das Kernproblem erweitert oder sogar geändert. Gleichzeitig findet eine Erweiterung der Software

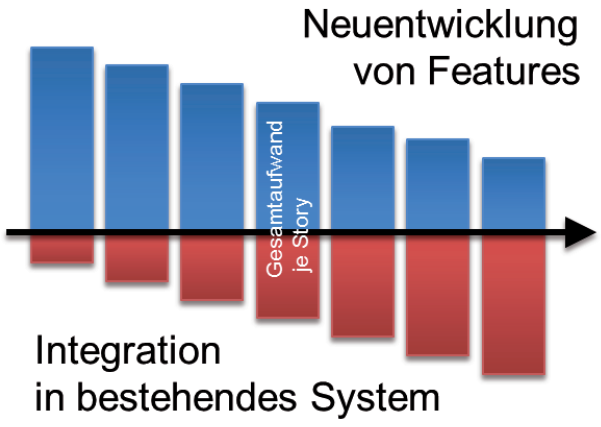


Abb. 1: Erosion der Agilität: Wachsender Anteil an Entwicklungsaufwand verschwindet in der Software.

statt, um auf diese geänderten Anforderungen zu reagieren. An dieser Stelle des Projektes wären Überlegungen zur geplanten Gesamtarchitektur des Systems angebracht, unterbleiben jedoch meist, was auf den Projektdruck in dieser Phase zurückzuführen ist.

Die Folge ist die Entstehung vieler Einzelkonzepte, die als Lösungen für Teilaspekte des geänderten Kernproblems der Software „hinzugefügt" werden. Die Metapher von „Balkonen" in der Software wurde in diesem Zusammenhang häufiger gebraucht.

Insgesamt steigt dadurch die Komplexität des Systems, da es kein greifbares Gesamtkonzept mehr gibt, sondern viele Teillösungen, die miteinander in Wechselwirkung stehen. Bei jeder Anpassung, die an der Software vorgenommen wird, müssen mehrere Konzepte angefasst und deren Verbindung untereinander verstanden und wiederhergestellt werden.

Die Erkenntnis, dass man sich in solch einer Situation befindet, kommt meist sehr spät. Ein Vorschlag, der dann häufig als Lösung gebracht wird, ist die komplette Renovierung oder Rekonstruktion des Systems. Damit sind sowohl hohe Kosten als auch große Risiken verbunden.

Trotzdem ist dieser Schritt vielfach notwendig, wenn die Gesamtarchitektur keine kontinuierlichen Verbesserungsmaßnahmen zulässt. Nach solch einer langwierigen Maßnahme steigt die Agilität der Entwicklung auch meist wieder an, neue Features können wieder schneller realisiert werden. Leider hält der Zustand nicht lange an, die Erosion von Architektur und damit auch von Agilität setzt schnell wieder ein.

Eine realistische, langfristige Lösung ist die kontinuierliche Investition in die Gesamtarchitektur des Systems, sodass diese während der Entwicklung ständig im Blick gehalten und an geänderte Anforderungen angepasst wird. An dieser Stelle wird oftmals die Frage geäußert, ob solch eine Strategie mit Agilität vereinbar ist. Eine schnelle Reaktion auf neue Anforderungen scheint schwer möglich, wenn dazu erst eine Adaption der Gesamtarchitektur erfolgen muss.

Kurzfristig betrachtet ist dieser Konflikt durchaus relevant, hat man jedoch den gesamten Lebenszyklus im Blick, so stehen die Investitionen in die Architektur des Systems in einem sehr positiven Verhältnis zu den Ersparnissen, die man damit während der Entwicklung erreichen kann. Diese sind unter anderem in der Reduktion der Komplexität des Systems begründet, was zu einer schnelleren Entwicklung führt.

Auf Basis einer geplanten Zielarchitektur des Systems sind genauere Abschätzungen von Stories möglich, denen gerade in agilen Prozessen eine hohe Bedeutung zukommt. Dies gilt explizit auch bei schnellen Änderungen des Backlogs, da auf Basis einer klaren Systemarchitektur die Umsetzung neuer Stories einfacher geplant werden kann.

Insgesamt zeigt sich, dass durch die Betrachtung von Architektur während der agilen Entwicklung die Agilität langfristig bewahrt werden kann. Im Weiteren soll dies durch die Erfahrungen aus einem aktuellen Projekt illustriert werden.

Kooperation von Industrie und Forschung

Das Ziel dieses Projektes ist die Neuentwicklung eines Business-Intelligence-Tools für den sogenannten „Document-Entry-Point". Es existierten keine Bestandskunden für das neue Produkt, jedoch ein bereits durch verwandte Produkte erschlossener Markt, der damit bedient werden soll. Um eine möglichst gute Ausrichtung auf die potenziellen Kunden zu haben, werden diese sehr früh in die Entwicklung einbezogen. In regelmäßigen Workshops werden deren Anforderungen erfasst und der aktuelle Stand des Produktes diskutiert. Daraus resultieren immer wieder neue User-Stories, die in das Backlog einfließen.

Das Entwicklungsteam, das für das Produkt verantwortlich war, entstand durch die Kooperation von Industrie und angewandter Forschung. In der Zusammenarbeit zwischen Entwicklern aus der Industrie und Forschern werden gemeinsam Praktiken aus dem Bereich Software-Engineering auf ihren praktischen Nutzen und ihre Anwendbarkeit überprüft. Insgesamt ergab sich daraus ein eher heterogenes Team.

Zum einen bestand es aus langjährigen Entwicklern, die sehr viel Programmiererfahrung ins Team mitbrachten, für die jedoch Software-Engineering-Praktiken eher ein Randthema darstellte. Auf der anderen Seite steuerten erfahrene Experten des Software-Engineering gezielt Wissen zu eben diesen Praktiken bei, besaßen aber selbst keine langjährige Entwicklungserfahrung in der Domäne der Dokumentverarbeitung. Durch Pairing und intensiven Austausch innerhalb des Teams wurde jedoch eine produktive Gruppe geschaffen, die auch flexibel genug war, um neue Wege zu gehen.

Zeitraubende Refactoring-Zyklen

Trotz der guten Zusammenarbeit im Team wurden bald die Probleme sichtbar, die anfangs beschrieben wurden. Die Geschwindigkeit der Feature-Entwicklung nahm ab, jede neue Story brauchte mehr Aufwand. Es wurde ein regelrechter Refactoring-Zyklus festgestellt, bei dem Teile des Systems mit jedem Sprint wieder umgebaut wurden. Die innere Komplexität der Software stieg immer weiter an, was am besten dadurch sichtbar wurde, dass ein immer größerer Teil des Aufwands, der für jede Story benötigt wurde, in das Erarbeiten eines Realisierungskonzeptes floss.

Das liegt darin begründet, dass dabei viele Einzelschritte notwendig waren, um zu einem tragfähigen Ergebnis zu kommen. Ein Großteil der Zeit nahm dabei das Verstehen des momentanen Systems in Anspruch, was durch die Vielzahl der Einzelkonzepte, die dort verwendet wurden, erschwert wurde.

Ein weiteres Konzept zu erarbeiten, das die neue Story unter Beachtung von vielen Mechanismen realisieren kann, gestaltete sich immer schwieriger. Die Stories wurden immer mehr zu kleinen Migrationsprojekten, bei denen durch viele Anpassungen die Realisierung eines neuen Features erfolgte. Eine Nebenerscheinung dieses Vorgehens waren Probleme bei der Integration, es traten ungeahnte Nebeneffekte auf, Teile des Systems verhielten sich zusammen anders als erwartet.

Insgesamt wurde intern eine hohe Komplexität wahrgenommen, obwohl die extern sichtbare Fachlichkeit doch eher überschaubar war. Dem Team fiel es immer schwerer den Aufwand für neue Stories zu vertreten, da der Aufwand für die „Integration" der neuen Features in das bestehende System von außen nicht gesehen wurde und entsprechend schwer darzustellen war.

Um aus diesem Zyklus zu entkommen wurden neue Ansätze getestet, die das Problem an der Wurzel packen sollten. Im Weiteren soll auf den erfolgreichsten dieser Ansätze eingegangen werden.

Verschiebung des Planungshorizontes

Am einfachsten lässt sich die gewählte Lö-

sung als Verschiebung des Planungshorizontes beschreiben, der im bisherigen Ansatz auf eine Story begrenzt war. Stattdessen werden nun alle Stories betrachtet, deren baldige Realisierung wahrscheinlich ist, was typischerweise alle Stories eines Epics sind.

Eine Art der Planung, die sich dabei als nützlich erwiesen hat, ist die Konzeption einer Systemarchitektur für diese Menge an User-Stories. Dabei ist der notwendige erste Schritt der Entwurf einer geeigneten fachlichen Architektur, die in der Lage ist, ein einheitliches konzeptionelles Modell bereitzustellen. Auf deren Basis kann die Lösung der User-Story beschrieben werden, ohne in technische Komplexität abzutauchen.

Auch Wechselwirkungen zwischen einzelnen Stories werden damit erkannt. Erst im nächsten Schritt wird die technische Umsetzung auf Basis des momentanen Systems geplant. Dabei findet eine gezielte Abwägung zwischen Umbau und Erweiterung von bestehenden Realisierungen statt. Nur so sind ganzheitliche Architekturentscheidungen möglich, die in ein einheitliches Konzept zur Umsetzung aller User-Stories eines Epics führen.

Um den Aufwand für diese Art der Planung gering zu halten, wird dabei mit einem Team von 2 bis 3 Personen auf einem hohen Abstraktionsniveau gearbeitet, das noch Entscheidungsfreiräume in der Implementierung lässt. Trotzdem enthält es alle grundlegenden Entscheidungen, sodass es möglich ist mittels Design-Walkthroughs potenzielle Fehler in der Konzeption frühzeitig zu entdecken.

Der entscheidende Unterschied zwischen Systemarchitekturen, die so vorab geplant werden und solchen, die iterativ während der Umsetzung mehrerer Stories entstehen, ist die Art, wie mit Gemeinsamkeiten und Wechselwirkungen zwischen Features umgegangen wird. Bei dem erweiterten Planungshorizont wird explizit Infrastruktur konzipiert, die die effiziente Umsetzung von mehreren Features ermöglicht. Sobald solche gemeinsam genutzten Konzepte erkannt werden, kann darauf auch in der Reihenfolge der Stories im Backlog eingegangen werden. Hierbei können solche Stories zuerst realisiert werden, die zum Aufbau einer gemeinsamen Infrastruktur beitragen.

Die konkrete Planung jedes Sprints bleibt nach wie vor erforderlich, gestaltet sich jedoch viel einfacher, da eine Orientierung an der Epic-Architektur erfolgen kann. Die Schwierigkeit besteht in dem Fall darin, eine geeignete Zerlegung des Gesamtkonzepts in realisierbare Einzelschritte zu finden.

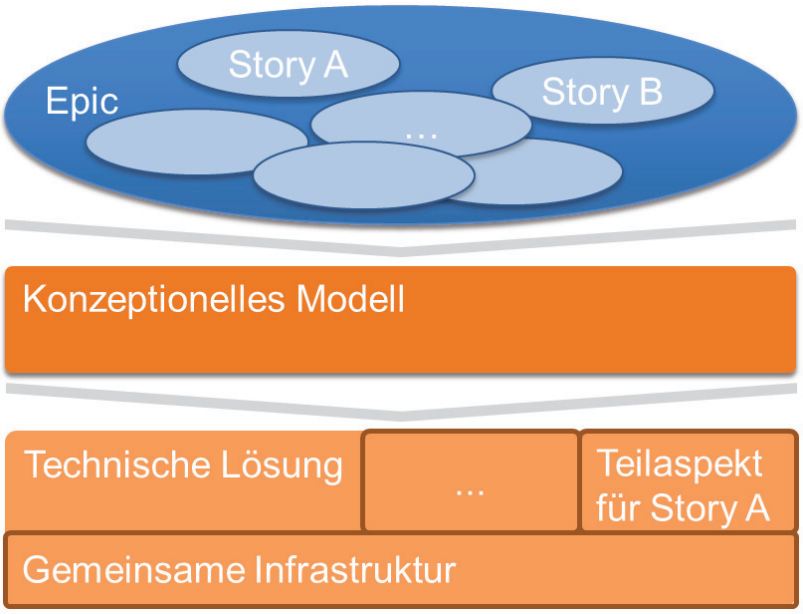


Abb. 2: Planung eines Epics.

Auf der einen Seite müssen alle relevanten Aspekte für die aktuellen Stories realisiert werden können, auf der anderen auch ein vertretbares Maß an zusätzlicher Infrastruktur, die eine Vorarbeit für zukünftige Stories darstellt. Dabei muss eine Abwägung zwischen „Bauvorleistung“ und späterer Einsparung von Refactorings getroffen werden. Nur durch diese Art der Planung ist es möglich, solch eine Entscheidung bewusst zu fällen und belastbare Aufwandsabschätzungen zu Rate zu ziehen.

Ein wichtiger Schritt, der nach jedem erfolgreichen Umsetzen einer Story erfolgen sollte, ist das Zurückspielen von neuen Erkenntnissen in die Planung. Damit ist gewährleistet, dass etwaige Abweichungen so gering wie möglich gehalten werden und das die Realisierung zukünftiger und bereits umgesetzter Stories konsistent bleibt.

Im bisherigen Verlauf des Projektes bestanden solche Feedback-Zyklen aus Ergänzungen zu den Konzepten, die in der Epic-Architektur vorgesehen sind. Potenziell wären auch größere Änderungen an der geplanten Architektur möglich, wobei solche Umbrüche genau abgewogen werden müssten, um nicht die Vorteile des einheitlichen Gesamtkonzeptes zu verlieren.

Agieren statt reagieren

Die Erfahrungen, die durch das Erweitern des Planungshorizontes hin zu Epic-Architekturen gemacht wurden, sind durchweg positiv – sowohl innerhalb des Entwicklungsteams als auch die extern wahrgenommenen Effekte. Innerhalb des Entwicklungsteams war vor allem die Reduktion des Refactoring-Aufwandes auffallend, der durch das Ausnutzen von Gemeinsamkeiten der Stories gewonnen werden konnte. Infrastrukturkomponenten werden ergänzt, anstatt sie immer wieder zu verändern oder sogar zu duplizieren.

Als ein Resultat daraus stieg die Zufriedenheit innerhalb des Teams an, was auch der nun verbesserten Möglichkeit zur Orientierung im Epic geschuldet ist. Jedem Teammitglied ist dabei klar, wie mit der momentanen Arbeit zur Erreichung des Epics beigetragen wird und wie offene Design-Entscheidungen getroffen werden müssen, um diesem Ziel näher zu kommen.

Extern wurde vor allem eine Steigerung der Umsetzungsgeschwindigkeit wahrgenommen. Die Zeit, die das Team braucht, um selbst eine bisher unbekannte Story des Epics zu planen und zu realisieren, ist wieder ihrer fachlichen Komplexität angemessen. Das frühzeitige Erfassen von Kundenfeedback wurde dadurch in kürzeren Zyklen möglich und die Entwicklung insgesamt als agil wahrgenommen.

Fazit

Inwieweit die positiven Effekte des Einführens von Epic-Architekturen nur den speziellen Projekteigenheiten geschuldet

sind wurde noch nicht abschließend evaluiert. Erste Erfahrungen aus anderen Projekten zeigen jedoch, dass diese durchaus auch in geändertem Kontext anwendbar sind.

Als weiterer interessanter Punkt wurde eine verbesserte Einbindung in unterschiedliche Entwicklungsprozesse identifiziert. Eine Integration in Scrum konnte in diesem Projekt bereits erfolgreich durchgeführt werden, für Kanban wurden erste Schritte in diese Richtung unternommen. Weiterhin besteht noch Potenzial in der Unterstützung der Planung selbst, also dem Erstellen der Epic-Architektur.

Die Auswahl der notwendigen Detailtiefe konnte zwar innerhalb dieses Projektes gut gewählt werden, jedoch fehlen noch allgemeine Regeln, die eine Abwägung zwischen benötigter Vorab-Investition und späterer Ersparnis beim Erstellen der Sprint-Planungen beachten. Innerhalb des hier beschriebenen Projektes wurde jegliche Dokumentation der Architektur als UML-Diagramme in Papierform vorgenommen. Dazu ist bereits geplant zu einem modellbasierten Ansatz überzugehen, der vor allem in größeren Projekten bessere Übersichtlichkeit ermöglicht.

Die wichtigste Erkenntnis, die in diesem agilen Entwicklungsprojekt gemacht wurde, ist der positive Effekt eines erweiterten Planungshorizontes auf die Agilität des Teams. Als angemessene Größe der Planung hat sich ein Epic bewährt, sodass eine signifikante Menge an User-Stories auf einmal betrachtet werden kann. Wir hoffen, dass solche Erfahrungen auch andere Teams dazu motivieren, aktiv die Entwicklung ihres Produktes zu steuern und damit langfristig ihre Agilität zu sichern. ■

Literatur

Brown, N., Nord, R., Ozkaya, I. (2010) Enabling Agility through Architecture, CrossTalk Nov/Dec 2010.
Bachmann, F., Nord, R., Ozkaya, I. (2012). Architectural Tactics to support rapid and agile stability, CrossTalk May/June 2012.