Agilität braucht Architektur!

Balthasar Weitzel[1], Matthias Naab[1], Mathias Scheffe[2]

[1] Fraunhofer IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern
{balthasar.weitzel, matthias.naab}@iese.fraunhofer.de
[2] Insiders Technologies, Brüsseler Straße 1, 67657 Kaiserslautern
m.scheffe@insiders-technologies.de

Abstract: Es hat sich gezeigt, dass ein agiler Ansatz, der nur durch 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 Software-Architektur 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 in einem R&D-Lab wurde gezeigt, wie solch ein Einsatz von Architektur gestaltet sein kann und welche Effekte damit zu erzielen sind. Hierbei wurde durch vorrausschauende Architektur-Arbeiten sowohl die Produktivität des Teams als auch der Entscheidungsspielraum des Product Owners erhöht. In diesem Beitrag beschreiben wir den konkreten Kontext des R&D-Labs von Insiders Technologies und Fraunhofer IESE und berichten von Erfahrungen des Transfers und der Anpassung von Architekturmethoden in ein agiles Entwicklungsprojekt.

1 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 Abbildung 1). Es hat sich gezeigt, dass der Verlust von Agilität durch die Erosion der Architektur verursacht wird. 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 initiales Konzept skizziert und zügig umgesetzt. Basierend auf den Erfahrungen, die dabei gemacht werden, wird das Kernproblem erweitert oder sogar

Diese Arbeit wurde unterstützt durch das Ministerium für Bildung und Forschung (BMBF) im Programm „Software Campus“, Förderkennzeichen 01IS12053

geändert. Gleichzeitig findet eine Erweiterung der Software statt, um auf diese geänderten Anforderungen zu reagieren. Während agile Entwicklungsprojekte die schrittweise Erweiterung der Anforderungen ausdrücklich akzeptieren und als Stärke ansehen, unterbleiben jedoch meist weitere Überlegungen zur Gesamtarchitektur. Einerseits werden agile Praktiken teils so interpretiert, dass nichts vorausgeplant wird, was nicht Bestandteil des aktuellen Sprints ist, andererseits verhindert Projektdruck die notwendige Investition. Die Folge ist die Entstehung vieler Einzelkonzepte, die als Lösungen für Teilaspekte des geänderten Kernproblems der Software „hinzugefügt" werden.

Abbildung 1: Erosion der Agilität

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. Während agile Entwicklung Refactorings als Mittel mitbringt, um konzeptionelle Inkonsistenzen zu vermeiden, ist es ab einem gewissen Punkt nicht mehr wirtschaftlich vertretbar jedes Refactoring durchzuführen. Durch die Betrachtung von Architektur während der agilen Entwicklung kann die Agilität langfristig bewahrt werden. Im Weiteren soll dies durch die Erfahrungen aus einem aktuellen Projekt illustriert werden.

2 Kooperation von Industrie und Forschung

Das Ziel dieses Projektes ist die Neuentwicklung eines Business-Intelligence-Tools für den sogenannten „Document-Entry-Point". Es existieren 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 neue User-Stories, die in das Backlog einfließen. Diese kurze Charakterisierung des Projekts kann als typisch für andere vergleichbare Projekte gesehen werden, die nach agilen Methoden durchgeführt werden.

Die Besonderheit dieses Projektes bestand in dem Umfeld, in dem es ausgeführt wurde. Das Entwicklungsteam, das für das Produkt verantwortlich war, entstand durch die Kooperation von Industrie und angewandter Forschung: In einem gemeinsamen Research & Development Lab überprüfen Fraunhofer IESE-Mitarbeiter zusammen mit Industriepartnern Praktiken aus dem Bereich Software Engineering auf ihren praktischen

Nutzen. Das eher heterogene Team bestand zum einen 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. Das Research & Development Lab wurde in dieser Konstellation über ein Jahr geführt, so dass auch langfristige Effekte untersucht werden konnten. Ziel ist es dabei, direkt anwendbare Lösung für solche Probleme zu finden, die in der industriellen Entwicklung häufig auftreten.

3 Zeitraubende Refactoring-Zyklen

Zu Beginn des Projektes wurde nach einem agilen Entwicklungsprozess nach Scrum [SS09] und TDD [B02] gearbeitet, der sich auf Wunsch der industriellen Entwickler relativ dogmatisch an agile Prinzipien hielt, wie zum Beispiel „jede Iteration muss direkten Kundennutzen bieten“. Trotz der guten Zusammenarbeit im Team wurden bald anfänglich beschriebene Probleme sichtbar: 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 in der Anzahl an Einzelschritten begründet, die 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, 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. Zielsetzung des R&D Labs ist es, genau solche Engineering Probleme gemeinsam zu analysieren, Gegenmaßnahmen zu erarbeiten und nachhaltig zu implementieren. Deshalb wurden neue Ansätze getestet, um das Problem der Agilitäts-Erosion zu beseitigen. Im Weiteren soll auf den erfolgreichsten dieser Ansätze eingegangen werden.

4 Verschiebung des Planungshorizontes

Am einfachsten lässt sich die gewählte Lösung 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.

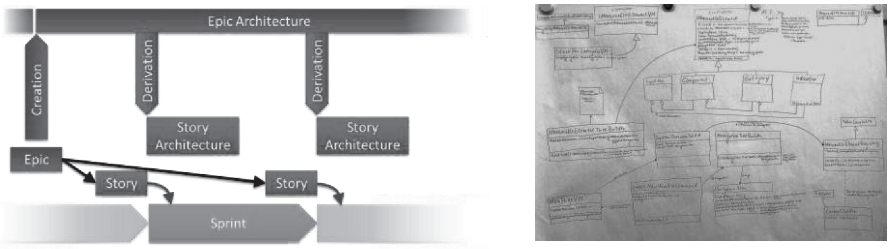

Abbildung 2: Bezug zwischen Epic- und Story-Architektur, Beispiel Epic-Architektur

Eine Art der Planung, die sich dabei als nützlich erwiesen hat, ist die Konzeption einer Systemarchitektur für diese Menge an User-Stories eines Epics. Diese Architektur bezeichneten wir als Epic-Architektur. Dabei ist der notwendige erste Schritt der Entwurf einer geeigneten fachlichen Architektur, die in der Lage ist, eine einheitliche Strukturierung des Problemraumes bereitzustellen. Auf deren Basis kann die fachliche Lösung der User-Story und Wechselwirkungen zwischen einzelnen Stories beschrieben werden, ohne in technische Komplexität abzutauchen. 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 die Epic-Architektur alle grundlegenden Entscheidungen, sodass es möglich ist, mittels Design-Walkthroughs potenzielle Fehler in der Konzeption frühzeitig zu entdecken. Als Art der Dokumentation wurde bewusst eine sehr einfache gewählt, die auch ein schnelles Ändern zulässt. Dabei wird bevorzugt am Whiteboard gearbeitet und UML-Diagramme mit zusätzlichen Notationen und Kommentaren erweitert. Der wichtige Aspekt dabei ist die Angemessenheit der Dokumentation für den Zweck, der in dem Fall das Schaffen eines einheitlichen Verständnisses bei allen Teammitgliedern ist. Ein Beispiel für solch eine Dokumentation ist in Abbildung 2 zu sehen. 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 unter fachlich ähnliche priorisierten

Stories solche zuerst realisiert werden, die zum Aufbau einer gemeinsamen Infrastruktur beitragen. Die konkrete Planung der Umsetzung der Stories eines Sprints bleibt nach wie vor erforderlich, gestaltet sich jedoch viel einfacher, da eine Orientierung an der Epic-Architektur erfolgen kann (siehe Abbildung 2). Die Ableitung eines konkreten Umsetzungsplanes einer Story erfolgt dann als Vorbereitung des Teams auf den nächsten Sprint, was in diesem Kontext als Story-Architektur bezeichnet wird. Dabei werden Details, die nicht in der Epic-Architektur vorgegeben sind, ergänzt. Hierbei handelt es sich um technische Realisierungsaspekte, die beispielsweise durch Prototypen erkannt wurden. Weiterhin findet eine Ableitung von konkreten Tasks statt, die dann im Team umgesetzt werden können.

Die Schwierigkeit besteht beim Ableiten von Story-Architekturen darin, eine geeignete Zerlegung des Gesamtkonzepts in realisierbare Einzelschritte zu finden. 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 Zusatzaufwand durch „Bauvorleistung“ und späterem gesparten Aufwand durch Entfallen 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 dass 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.

5 Günstig agieren statt teuer 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. Die Verbesserungen in exakte Zahlen zu fassen ist eher schwierig, da innerhalb des Projektes keine Kontrollgruppe aufgesetzt werden konnte, die ohne unseren Ansatz gearbeitet hat. Basierend auf einem Vergleich des Aufwandes für Stories mit einer vergleichbaren Komplexität vor und nach der Einführung des Konzeptes der Epic-Architektur kann eine Reduktion von 25% des Aufwandes festgestellt werden. Das führt zu einer Ersparnis von ungefähr 0,75 Tagen für das gesamte Team. Im Gegensatz dazu steht der Aufwand zur Erstellung einer Epic-Architektur, die zwei Personen für 1,5 Tage beschäftigt hat. Bei der Größe der im Projekt verwendeten Epics von 15 Stories ergeben sich daraus 0,1 Tage für zwei Personen in einer Story, die für die Erstellung der

Epic-Architektur verwendet wurden. Im Vergleich ergibt sich daraus eine deutliche Ersparnis.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.

6 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. 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.

7 Literaturverzeichnis

[BNO10] Brown, N., Nord, R., Ozkaya, I. (2010) Enabling Agility through Architecture, CrossTalk Nov/Dec 2010.

[BNO12] Bachmann, F., Nord, R., Ozkaya, I. (2012). Architectural Tactics to support rapid and agile stability, CrossTalk May/June 2012.

[SS09] Schwaber K., Sutherland J. (2009) Scrum Guide, Scrum Alliance, vol. 19, no. 6, p. 21.

[B02] Beck K. (2002) Test Driven Development: By Example, Addison-Wesley Professional, p. 192.