SCRUM UND ARCHITEKTUR: KONZEPTIONELLE INTEGRITÄT IM SCRUM-PROZESS

Architekturentscheidungen kooperativ herbeizuführen und erst dann zu treffen, wenn sie benötigt werden, ist ein qualitätsverbesserndes Paradigma der agilen Softwareentwicklung. Konzeptionelle Integrität wird für die Software aber nur dann nachhaltig erreicht, wenn eine Person für die Architektur verantwortlich ist. Das geht nicht ohne Architekten, deren Rechte und Pflichten zur Anwendung im Scrum-Prozess in diesem Artikel erläutert werden.

▶ der autor

Ulf Schneider
(E-Mail: us@datenlabor.net)
ist Geschäftsführer der Datenlabor GmbH. Er war mehr als zehn Jahre bei IBM Global Services als IT-Architekt in Softwareentwicklungsprojekten tätig und ist Certified Scrum Professional.

Scrum ist teamorientiert und unterteilt das Entwicklungsvorgehen in kurze Iterationen. Es stellt sich die Frage, wie bei der Softwareentwicklung mit Scrum ein konzeptionell integrer Produktentwurf erreicht wird: ein Entwurf, der für die Benutzer, Entwickler und Betreiber der Software intuitiv verständlich ist und der selbst nach vielen Auslieferungsiterationen, in denen das Team sich auf den Mikrokosmos der Probleme der jeweils aktuellen Iteration konzentriert, ganzheitlich sinnvoll und klar bleibt.

Der Entwurf eines Softwareprodukts manifestiert sich in der Systemarchitektur. Soll konzeptionelle Integrität erreicht werden, muss diese in der Architektur berücksichtigt werden. Vor diesem Hintergrund mache ich im Folgenden organisatorische und handwerkliche Vorschläge, die alle das Ziel haben, die Architekturarbeit im Scrum zu unterstützen. Dabei hebe ich insbesondere die Verantwortung des Architekten für die konzeptionelle Integrität des Produkts hervor.

Scrum und Architektur

Die agile Softwareentwicklung erkennt Veränderlichkeit als eigenständigen Wert an: „Responding to change over following a plan." (vgl. [Bec01]). Demnach ist es sinnvoll, das eigene Verhalten zur Erreichung von Ergebnissen sowie die Ergebnisse selbst kontinuierlich zu beobachten, um in der Folge durch Lerneffekte und Verhaltensanpassungen bessere und aufgabenangemessene Ergebnisse zu erzielen. Scrum ist ein Management-Framework für die agile Produktentwicklung und unterstützt diese Form der Beobachtung und Anpassung durch die Einteilung der Arbeit in ein- bis vierwöchige Iterationen, verbunden mit einer Produkt- und Prozessinspektion am Ende jeder Iteration.

Für die Erbringung von Innovationsleistungen, wie z. B. in Softwareentwicklungsprojekten, wird damit dem Umstand Rechnung getragen, dass jeder Innovationsprozess ein Lernprozess ist und somit eine kontinuierliche Verhaltensanpassung erfordert. Hierzu gehört es, Architektur- und Designentscheidungen so spät wie möglich zu treffen, um den Erkenntnisgewinn, der mit dem Fortschreiten des Innovationsprozesses verknüpft ist, in die Entscheidung einfließen zu lassen. Desweiteren wird so früh wie möglich funktionsfähige Software ausgeliefert, um endlose Diskussionen zu vermeiden und das Momentum erledigter Arbeit auszunutzen: Eine Herangehensweise, die als „Deliver early and decide late" bezeichnet wird. Ken Schwaber und Mike Beedle prägten in diesem Zusammenhang den Ausdruck: „Cut through the noise by taking action" (vgl. [Sch04]).

Architekturentscheidungen werden auf der strategischen Ebene getroffen und haben weitreichende Auswirkungen auf die auch von Benutzern wahrgenommene Gestaltung des Softwareprodukts. Diese Entscheidungen definieren den Lösungsraum zu einer gegebenen Problemstellung und stellen insofern die Brücke zwischen den Anforderungen und der Umsetzung her. Es ist die Architektur, die es erlaubt, ein Softwaresystem mit vertretbarem Aufwand an veränderliche Anforderungen anzupassen (vgl. [Sie04]).

Architekturentscheidungen werden unter den folgenden Prämissen (vgl. [Fri10]) getroffen:

- *Vertretung der Interessen* aller Stakeholder am System über den gesamten Lebenszyklus
- *Minimierung der Gesamtkosten* für das System über den gesamten Lebenszyklus

Designentscheidungen werden dagegen auf taktischer und operativer Ebene getroffen und beeinflussen in hohem Maße die Codequalität des Softwareprodukts. Sie strukturieren den Lösungsraum, der durch die Architekturentscheidungen aufgespannt wird. Designentscheidungen werden nicht unbedingt von den Benutzern des Softwareprodukts wahrgenommen – es sei denn, es handelt sich um Entscheidungen, die die Benutzungsschnittstelle betreffen. Der Übergang zwischen Architektur- und Designebene kann fließend sein (siehe Abbildung 1).

Scrum geht von einer kooperativen Arbeitsorganisation aus. Mit Ausnahme des *Product Owners* und des *Scrum Masters* sind keine Rollen für ein Scrum-Projekt benannt. Es wird angenommen, dass die Teammitglieder weitere Rollen wahrnehmen und durch Selbstorganisation zu aufgabenangemessenen Lösungen für ihre Probleme gelangen.

Wie aber werden die Architekturentscheidungen im Team getroffen? Sind sie

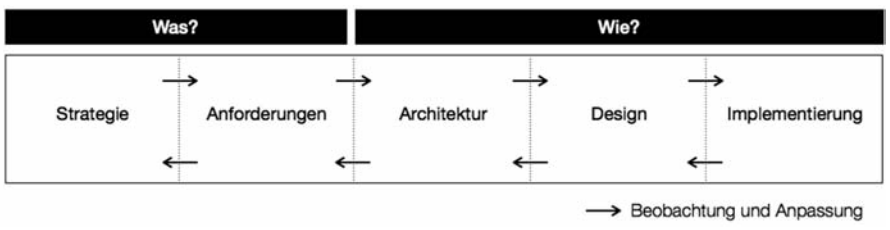

Abb. 1: Architektur als Brücke zwischen Anforderungen und Umsetzung.

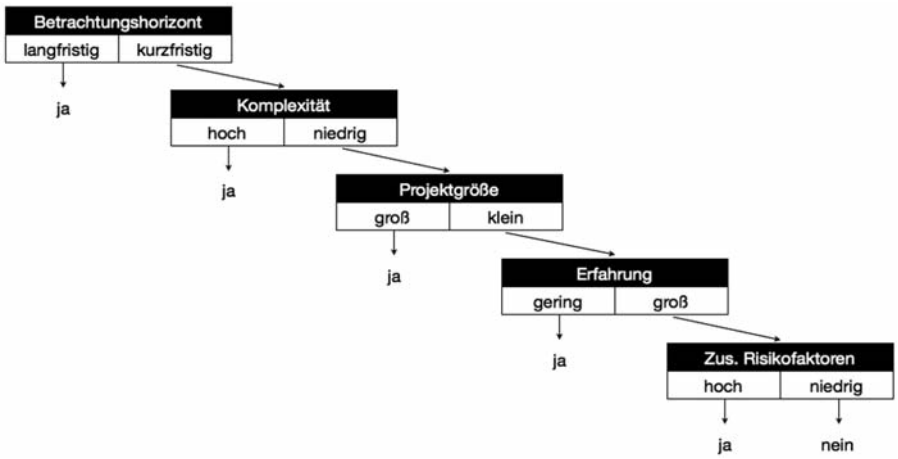

Abb. 2: Entscheidungsbaum für den Einsatz eines Architekten (nach [Fri10]).

Folge eines teamorientierten Brainstormings und ist die Entscheidung „plötzlich klar", solange der Zeitpunkt der Entscheidung möglichst spät liegt? Wie wird sichergestellt, dass die getroffenen Entscheidungen einem durchgängigen Konzept gehorchen, also zu konzeptioneller Integrität führen? Ich vertrete die These, dass die kooperative Herleitung von Architekturentscheidungen zwar qualitätsverbessernd ist und verfolgt werden sollte, dass konzeptionelle Integrität aber nur dann nachhaltig erreicht werden kann, wenn die finale Entscheidungskompetenz für architektonische Fragen in den Händen einer oder höchstens zweier eng zusammenarbeitender Personen liegt.

Prozess und Mensch

Eine gute Architektur führt zu konzeptioneller Integrität. Kann eine gute Architektur durch organisatorische Maßnahmen, wie Prozesse und Kommunikationsstrukturen, erreicht werden oder ist hierfür die Expertise eines erfahrenen und inspirierten Architekten wichtiger?

Aus der Perspektive der agilen Softwareentwicklung mit Scrum kommt man schnell zu einer Antwort. Die erste Wertaussage des Agilen Manifests macht klar, dass Menschen und ihre Interaktionen einen höheren Stellenwert haben als Prozesse und Werkzeuge (vgl. [Bec01]). In einem agilen Umfeld hat ein Architekt also immer einen höheren Stellenwert als ein Architekturprozess. Dabei bietet uns Uwe Friedrichsen mit seinem Entscheidungsbaum für den Einsatz eines Architekten eine gute Hilfestellung zur Beantwortung der Frage, ob ein Architekt für ein Vorhaben benötigt wird (siehe Abbildung 2).

Auch aus einem anderen Blickwinkel ist es sinnvoll, der Person den Vorzug vor dem Prozess zu geben. Softwareentwicklung fordert kommunikativ-soziale Fertigkeiten, die ingenieurmäßige Beherrschung von Fertigungspraktiken und Kreativität. Für ein innovatives Produkt ist Kreativität neben den beiden zuvor genannten Fertigkeiten eine wesentliche Lösungskomponente. Menschen sind kreativ, nicht aber Prozesse. Kreativität lässt sich nicht durch organisatorische Strukturen herbeiführen, sondern nur unterstützen. Damit sollte kreativen Menschen allgemein und kreativen Architekten speziell, ein hoher Stellenwert im Rahmen der Softwareentwicklung eingeräumt werden.

Das bedeutet nicht, dass die organisatorische Gestaltung des Arbeitsfeldes ausgeklammert werden kann. Organisation hat die Aufgabe, den kommunikativ-sozialen Aspekt der Softwareentwicklung effizient zu gestalten – dabei sind die geschaffenen Kommunikationsstrukturen prägend für das Produkt. „Conways Law" (vgl. [Con68]) liefert uns den wesentlichen Blickwinkel: „Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure." Die Kommunikationsstrukturen des Projekts finden sich im Design des Produkts wieder.

Wenn die zentrale Rolle eines Architekten für die konzeptionelle Integrität des Produkts anerkannt wird, darf der organisatorische Kontext dem nicht zuwider laufen. Eine komplexe Organisation mit vielen Mitarbeitern führt zu einem komplexen und aufgeblähten Produkt, eine schlanke Organisation mit kompetenten Mitarbeitern führt zu einem fokussierten und schlanken Produkt.

Scrum bietet uns mit seiner reduzierten und an Ergebnissen ausgerichteten, teamorientierten Organisation ein adäquates Hilfsmittel, um fokussierte und schlanke Produkte zu erstellen. Der Architekt trägt im Scrum-Projekt dazu bei, die konzeptionelle Integrität des Produkts aufrechtzuerhalten.

Der Architekt im Scrum

Die nachfolgenden Vorschläge zur organisatorischen Eingliederung des Architekten in ein Entwicklungsvorhaben hat Frederick P. Brooks in „The Mythical Man Month" (vgl. [Bro95]) und „The Design Of Design" (vgl. [Bro10]) vorgestellt. Zur Anwendung im Scrum-Prozess möchte ich hier Brooks Vorschläge ergänzen:

- *Eingliederung:* Die Eingliederung des Architekten in das Team oder seine Separatstellung analog zum *Product Owner* hängt von der Größe eines Vorhabens ab. Wird lediglich ein Scrum-Team benötigt, nimmt der Architekt eine Rolle innerhalb des Teams wahr. Benötigt man dagegen mehrere Scrum-Teams, nimmt der Architekt eine separate Rolle wahr.
- *Entscheidungskompetenz:* Analog zur Entscheidungskompetenz des *Product Owners* bei geschäftlichen Fragestellungen, hat der Architekt das letzte Wort bei Architekturentscheidungen.
- *Der Architekt gehört zum Projekt:* Er ist proaktiv für die Gestaltung des Lösungsraums und ebenso wie jedes Teammitglied für die Verbesserung des Entwicklungsprozesses verantwortlich. Von ihm erwartet man dieselbe Kooperationsbereitschaft und -fähigkeit wie von allen anderen Teammitgliedern.
- *Der Architekt kann programmieren:* Der Architekt ist im Scrum-Team anerkannt. Das gilt für die Techniker ebenso wie für die Vertreter der Fachbereiche. Sein Beitrag wird als Bereicherung für die Schaffung eines erfolgreichen Produkts wahrgenommen und nicht als Bevormundung. Der Architekt stiftet in seinem Projekt Nutzen. In diesem Zusammenhang ist es für Softwareprojekte unerlässlich, dass der Architekt programmieren kann und in der Lage ist, sich in der Diskussion mit Programmierern und

z. B. Datenbankadministratoren auch in technische Details hineinzuversetzen. Nur so kann er die Tragweite seiner Entscheidungen verstehen und verantworten. In den meisten Fällen ist der professionelle Hintergrund eines Architekten technisch geprägt. Das fordert von ihm besondere Offenheit sowie Lern- und Kommunikationsbereitschaft in Bezug auf die Fachdomäne seines Projekts.

- *Erster unter Gleichen:* In umfangreichen Projekten mit mehreren Scrum-Teams kann der Architekt einen oder mehrere Assistenten haben, die einzelnen Scrum-Teams zugeordnet sind und dort konkret mit ihrem Team arbeiten. Dieses Szenario kann beispielsweiseTeil eines Ausbildungsprogramms für die Assistenzarchitekten sein, die auf diesem Weg in konkreten Projekten Erfahrungen sammeln und Verantwortung tragen, dabei aber auf die Expertise und Seniorität eines erfahrenen Architekten zurückgreifen können. Auch wenn der Ausbildungsaspekt nachrangig ist – wenn die beteiligten Architekten einen vergleichbar hohen Kenntnisstand haben, sich also gegenseitig ergänzen – muss für das Scrum-Projekt klar sein, welcher Architekt die Rolle des „Ersten unter Gleichen" hat.

Die genannten Punkte fordern viel von einem Architekten – allerdings gilt das für alle Beteiligten eines Scrum- oder IT-Projekts. Von einem Entwickler wird erwartet, dass er auf dem Stand der Technik entwickeln und programmieren kann, von einem *Product Owner* erwartet man, dass er in seiner Organisation wirkt und über eine hohe geschäftliche Entscheidungskompetenz verfügt, und von einem Fachexperten erwartet man die Erstellung schlüssiger fachlicher Vorgaben und Testfälle. Entsprechend sollte auch der Architekt seine Rolle ausfüllen bzw. sich durch Beobachtung und Anpassung im Laufe des Projekts entsprechend kontinuierlich weiterentwickeln können.

Architekturgremien

Unternehmensweite Architekturgremien, die mit dem Entscheidungsbedarf aus verschiedenen Projekten versorgt werden und Freigaben, Sperren oder Modifikationen von Architekturentscheidungen vornehmen, stellen häufig sowohl für die Unternehmens- als auch für die Projektarchitektur einen Flaschenhals dar. Man blockiert sich gegenseitig auf der Suche nach Synergieeffekten, und Verantwortung aus dem Projekt wird auf das übergeordnete Gremium verlagert. Durch zu viele Beteiligte mit unterschiedlichen Interessen sind die getroffenen Entscheidungen oftmals nicht fokussiert und für keines der betroffenen Projekte optimal.

Ein unternehmensweites Architekturgremium sollte Randbedingungen und strategische Architekturziele aus der Perspektive des Unternehmens erarbeiten und sich so wenig wie möglich an der konkreten Projektarbeit beteiligen – es sei denn, der Projektarchitekt wünscht dies. Um praxisrelevante Ergebnisse zu erzielen, ist es sinnvoll, wenn dieselben Architekten, die in Projekten tätig sind, auch im unternehmensweiten Architekturgremium arbeiten. Dabei sind die verschiedenen Zielsetzungen der Gremienarbeit und der Projektarbeit aber klar voneinander zu unterscheiden.

Der Projektarchitekt muss die aus dem unternehmensweiten Gremium vorgegebenen Randbedingungen einhalten und die geforderten Architekturziele (auf Projekt- und Unternehmensebene) erfüllen. In diesem vorgegebenen Rahmen führt er Architekturentscheidungen zügig – nach dem Bedarf seines Projekts und in Abstimmung mit seinem Projektteam – herbei. Der Architekt verantwortet seine Entscheidungen – dazu gehört auch die Entscheidung, wann er ein unternehmensweites Architekturgremium hinzuzieht und wann nicht.

Vermeiden Sie nach Möglichkeit Design durch Gremien. Das konzeptionelle Design liegt in der Hand des Architekten und ist nicht Aufgabe eines Architekturgremiums. Das Design durch ein Gremium ist die Garantie für ein aufgeblähtes Produkt.

Werkzeuge

Im Folgenden stelle ich konkrete Hilfsmittel vor, die der Architekt in Kooperation mit dem Team für die Architekturarbeit einsetzen kann. Zum Teil sind die vorgestellten Dokumenttypen und Praktiken als klassische Architekturwerkzeuge bekannt, sodass ich an dieser Stelle ergänzende Vorschläge zur Einbettung in den Scrum-Prozess mache.

Produktvision

Die Produktvision (vgl. [Pic09]) dient in Scrum zur Artikulation der strategischen Ziele des Vorhabens. Sind im Projektverlauf Entscheidungen zu treffen, Güteabwägungen anzustellen und Anforderungen abzuleiten, gibt die Produktvision hier Orientierung. Insbesondere sollte keine Entscheidung getroffen werden, die den strategischen Zielen der Produktvision zuwiderläuft.

Häufig sind mit umfangreichen Vorhaben unausgesprochen auch architektonische oder organisatorische Ziele verknüpft. Architekturziele sind funktionale („Was soll das System leisten?") oder nichtfunktionale („Wie gut soll das System die Leistung erbringen?") Anforderungen an das zu erstellende System und haben per Definition einen strategischen Charakter. Der Architekt hat ein starkes Interesse, die Architekturziele eines Vorhabens von vornherein zu identifizieren, diese mit den Stakeholdern abzustimmen und in der Produktvision zu verankern. Lassen Sie sich als Architekt auf kein Projekt ein, wenn diese Arbeit nicht getan werden kann, anderenfalls sind die Chancen für einen Projekterfolg gering. Stimmen Sie sich bei der Formulierung der Architekturziele eng mit dem *Product Owner* ab. Es gilt nach Johann Wolfgang von Goethe das Motto: „Wer das erste Knopfloch verfehlt, kommt mit dem Zuknöpfen nicht zu Rande."

Wie zentral die Formulierung und Kenntnis der Architekturziele ist, verdeutlichen die **Abbildungen 3, 4 und 5**, die verschiedene Architekturziele verinnerlichen (nach [Pre10]).

Randbedingungen

Technische oder organisatorische Randbedingungen gibt es in jedem Projekt. Randbedingungen schränken den Lösungsraum ein, der durch den Architekten aufgespannt wird. Gleichzeitig bieten Randbedingungen aber auch Orientierung und Verankerungspunkte für die Lösungsarchitektur.

Randbedingungen sind beispielsweise gegeben durch die Vorgabe, einen unternehmensweiten Verzeichnisdienst zur Verwaltung von Benutzerdaten zu verwenden, durch ein bestimmtes Zeitfenster für die Stapelverarbeitung von Datensätzen oder durch die Definition von *Queueing*-Schnittstellen zu bestehenden Systemen. Auch eine Kostenobergrenze ist eine Randbedingung (vgl. [arc42]).

Eine Liste der Randbedingungen sollte vom Architekten geführt werden, um sich und anderen diese Grenzen bewusst zu

Abb. 3: Architekturziel „Repräsentation" (Das Foto zeigt die Villa La Rotenda in Vicenza. Quelle: http://upload.wikimedia.org/wikipedia/commons/e/ea/La_Rotonda.jpg*).*

Abb. 4: Architekturziel „Verteidigung" (Das Foto zeigt die Festung Hohensalzburg. Foto: Andre Bossi, siehe: http://upload.wikimedia.org/wikipedia/commons/7/7e/2194_-_Salzburg_-_Festung_Hohensalzburg_from_Richterhöhe.JPG*).*

Abb. 5: Architekturziel „Mobilität" (Das Foto zeigt ein Aquarell von Karl Bodmer, siehe: http://upload.wikimedia.org/wikipedia/commons/d/df/Tipi01.jpg*).*

machen. Für die Ermittlung der begrenzenden Ressource (siehe unten) kann die Liste der Randbedingungen ebenfalls hilfreich sein. Gegebenenfalls können die Randbedingungen auch als Bestandteil der Produktvision dokumentiert werden.

Kontextdiagramm

Das Kontextdiagramm hilft, das Umfeld des Systems zu verstehen. Das System wird dabei als Black Box betrachtet. Es geht darum, die von außen auf das System wirkenden Kräfte mit ihren Ein- und Ausgabeschnittstellen zu identifizieren,

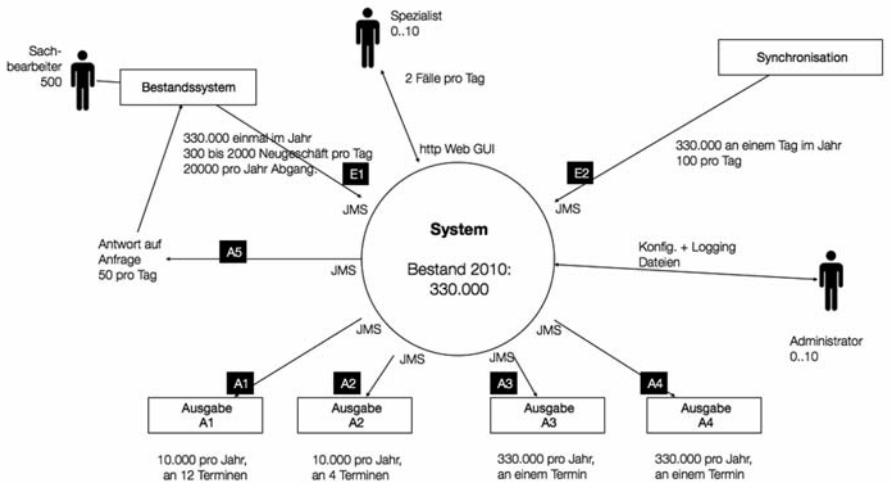

Abb. 6: Das Kontextdiagramm entstammt einem echten Projekt (die Namen der Schnittstellen sind anonymisiert). Verwenden Sie in Ihrem Projekt konkrete Schnittstellenbezeichner, aus denen erkenntlich ist, welche Funktion die Schnittstellen erfüllen.

Systemgrenzen und Verantwortlichkeiten zu erkennen und Risiken sichtbar zu machen.

Das System wird als Kreis in der Mitte des Kontextdiagramms dargestellt. Sämtliche darauf einwirkende Systeme werden als Rechtecke und Personen als Strichmännchen mit gerichteten Kanten (für Ein- und Ausgabedatenströme) um das System positioniert. Nicht-funktionale Anforderungen, wie z. B. die Anzahl von Datensätzen, die in einer bestimmten Zeit über eine Kante bewegt werden, können als Gewichte an den Kanten oder Systemrechtecken notiert werden (siehe Abbildung 6).

Das Kontextdiagramm wird bereits in einer frühen Projektphase erstellt und kann sehr gewinnbringend als Gruppenarbeit an einer Metaplanwand entwickelt werden. Es kann hilfreich sein, nicht nur um das interdisziplinär besetzte Scrum-Team, sondern auch weitere Stakeholder hinzuzuziehen. Durch die Gruppenarbeit lassen sich zügig verschiedene Perspektiven auf das System ermitteln und kommunizieren. Das System wird dabei in der Mitte einer Metaplanwand mit einer Moderationskarte dargestellt. Die darauf einwirkenden Systeme und Personen werden ebenfalls mit Moderationskarten abgebildet. Datenflüsse zwischen dem zu erstellenden System und den darauf einwirkenden Systemen und Personen werden als Linien eingezeichnet.

Story-Board

In Scrum werden Anforderungen in Form von User-Storys erfasst. Ein *Product Backlog* wird als priorisierte Liste dieser User-Storys geführt.

Das Story-Board stellt – zusätzlich zur listenorientierten Darstellung des *Product Backlog* – die wesentlichen User-Storys in einem Zusammenhang dar, der die Funktionsweise des Systems verdeutlicht. Anders als in der Kontextsicht, die eine Black-Box-Betrachtung ist, geht es beim Story-Board um eine White-Box-Betrachtung, bei der gerade systeminterne Zusammenhänge hervorgehoben werden sollen.

Das führt zu einer komponentenorientierten Anordnung und ebensolcher Betrachtung der User-Storys. Umfangreiche User-Storys, die auf oberster Ebene als wertschöpfende *End-to-End*-Beschreibung einer gewünschten Funktionalität dokumentiert sind, können in kleinere Bestandteile (kleinere User-Storys oder Aufgaben zur Umsetzung der Storys) zerlegt und im Story-Board komponentenorientiert angeordnet werden. Für Projektphasen, in denen infrastrukturelle Arbeiten im Vordergrund stehen, ist das Story-Board (siehe Abbildung 7) gut nutzbar.

Die Darstellung hilft allen Beteiligten, den konzeptionellen Zusammenhang der Storys und Aufgaben zu erkennen, und unterstützt den Anspruch der Architektur, ein Mittler zwischen Anforderungen und Umsetzung zu sein. Das Story-Board verbessert das Verständnis der Umsetzungsstruktur und das Mapping der Anforderungen auf diese Struktur für alle Beteiligten.

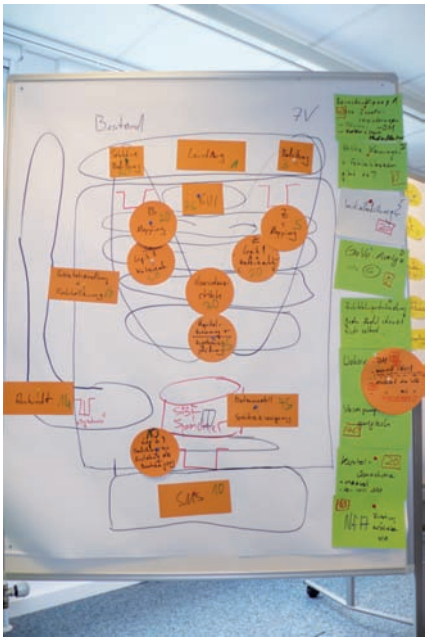

Abb. 7: Die initiale Fassung eines im Team erarbeiteten Story-Boards.

Das Team sollte das Story-Board zu einem frühen Zeitpunkt mit Hilfe einer Metaplanwand, auf der die User-Storys und Aufgaben angeordnet und um skizzierte Beziehungen zueinander erweitert werden, erarbeiten. Die Form der Darstellung ist frei, eine Orientierungshilfe bieten die in Abbildung 7 und 8 dargestellten Story-Boards.

Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen haben häufig systemübergreifenden Charakter und sind damit nicht nur einer einzelnen Systemkomponente oder User-Story zuzuordnen. Strategische Architekturziele, die sich aus den nicht-funktionalen Anforderungen ergeben, sind in der Produktvision zu behandeln (siehe oben).

Es gibt jedoch auch nicht-funktionale Anforderungen, die vielleicht keinen Eingang in die Produktvision finden, aber dennoch Einfluss auf die Systemgestaltung haben, wie z. B. Anforderungen an Zuverlässigkeit, Leistung und Effizienz, Sicherheit, Skalierbarkeit, Verfügbarkeit, Änderbarkeit, Testbarkeit und Bedienbarkeit. Aufgrund des systemübergreifenden Charakters dieser Anforderungen ist es Aufgabe des Architekten, diese Anforderungen aus einer ganzheitlichen Perspektive zu ermitteln und zu dokumentieren. Als guter Einstiegspunkt hierfür hat sich das beschriebene Kontextdiagramm erwiesen.

Die nicht-funktionalen Anforderungen sollten in einem übergreifenden Dokument zusammengefasst werden. Eine Berücksichtigung von nicht-funktionalen Anforderungen im *Product Backlog* bietet sich nur dann an, wenn sich diese Anforderungen direkt einer User-Story zuordnen lassen. In diesem Fall kann die User-Story um entsprechende Akzeptanzkriterien erweitert werden.

Wo es möglich ist, sollte der Architekt die nicht-funktionalen Anforderungen in messbare Qualitätsattribute transformieren, um deren Einhaltung oder Verfehlung überprüfbar zu machen und um die Ausrichtung aller Akteure auf die Erreichung der quantifizierten Anforderung zu erleichtern. Speziell für das weiter unten dargestellte Verfahren der begrenzenden Ressource ist eine derartige Quantifizierung erforderlich.

Es kann auch sinnvoll sein, die in messbare Attribute transformierten, nicht-funktionalen Anforderungen in die *Definition of Done* aufzunehmen. Die messbar gemachten Anforderungen sollten in Tests münden und bei jedem Build überprüft werden.

Architekturentscheidungen

Für den Architekten repräsentieren die Architekturentscheidungen die bewusste Festlegung des Lösungsraums für das zu erstellende System. Architekturentscheidungen sind Verankerungspunkte für weitere, sich daraus ableitende Entscheidungen. Damit sind Architekturentscheidungen im Projektverlauf schwer zurückzunehmen. Mit Stefan Zörner (vgl. [Zör10]) kann man sagen: „Wer diese Entscheidungen nachvollziehbar herleitet und verantwortet, der entwirft die Architektur."

Alle bisher vorgestellten Dokumenttypen beeinflussen die Art und Weise, in der Architekturentscheidungen getroffen werden.

Um eine Softwarearchitektur umzusetzen, müssen getroffene Entscheidungen kommuniziert werden. Neben der verbalen Kommunikation, die zentraler Bestandteil von Scrum ist, gehört dazu die Verschriftlichung der Entscheidungen. Das gesamte Team wird in die Lage versetzt, auch (lange) zurückliegende Entscheidungen nachzuvollziehen. Stefan Zörner hat einen praxisnahen Vorschlag zur konkreten Gestaltung von Dokumenten für Architekturentscheidungen gemacht, der in Abbildung 9 aufgegriffen wird.

Begrenzende Ressource

Die nicht-funktionalen Anforderungen, Randbedingungen und Architekturziele

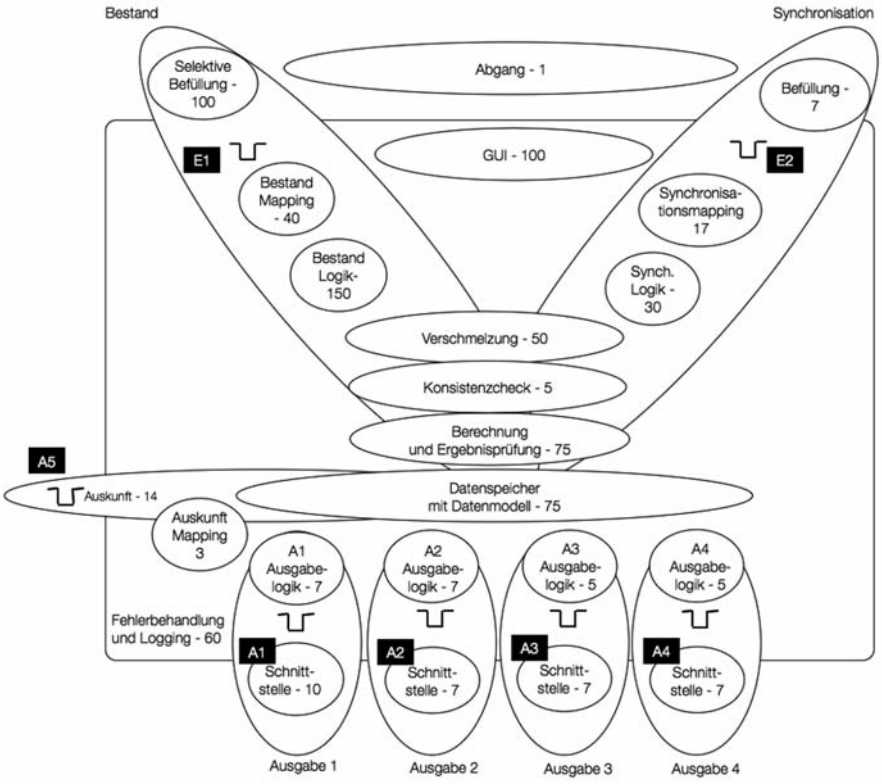

Abb. 8: Das elektronisch erfasste Story-Board nach einigen Erweiterungsschritten. Dieses anonymisierte Story-Board entstammt einem echten Projekt.

Abb. 9: Aufbau einer Architekturentscheidung (nach [Zör09]).

führen dazu, dass beim Entwurf eines Systems bestimmte Ressourcen knapp werden. Bereits in frühen Entwicklungsphasen sollte man diese knappen Ressourcen identifizieren und relevante Grenzwerte ableiten. Die Grenzwerte sind in Bezug zu den aktuell durch das Team erreichten Werten zu setzen. Hier zwei Beispiele:

- *Grenzwert:* Mindestens 250.000 Datensätze pro Stunde müssen verarbeitet werden.
- *Zurzeit erreichter Wert:* 180.000 Datensätze werden pro Stunde verarbeitet.

Eine tägliche Überprüfung der Zielerreichung ist anzustreben. Die frühe und regelmäßige Kommunikation des Zielerreichungsgrades führt zur Ausrichtung des Scrum-Teams auf dieses Ziel und zur Optimierung des Systems hin zur Einhaltung des Grenzwertes. Die wichtigste Aufgabe für den Architekten ist es, gemeinsam mit dem Team die wesentliche begrenzende Ressource festzustellen und Maßnahmen zur Messung der Zielerreichung zu etablieren.

End-to-End Skeleton

In umfangreichen Entwicklungsvorhaben mit mehreren komponentenorientierten Scrum-Teams kann ein integriertes Vorgehen durch Verwendung des *End-to-End-Skeletons* erreicht werden. Dabei werden alle Komponenten des Systems von Beginn an integriert getestet. Wesentlich ist die Definition aller Schnittstellen zwischen den Komponenten. Diejenigen Komponenten, die erst in der Zukunft programmiert werden, werden durch so genannte *Mocks*[1]) repräsentiert. Das sind Programmobjekte ohne Geschäftslogik, die lediglich die geforderten Schnittstellen und den benötigten Code zum Durchlaufen des Tests implementieren.

Das Ziel des *End-to-End-Skeleton* ist der durchgehende Datenfluss über alle Komponenten. Damit wird früh im Entwicklungsprozess eine integrierte Sicht auf das Gesamtsystem gefordert und die Entwicklungsarbeit richtet sich nach konkreten Schnittstellen aus. Die beteiligten Teams arbeiten dadurch beizeiten mit einer ganzheitlichen Sichtweise. Kann kein durchgehender Datenfluss erreicht werden, wird dies sofort sichtbar.

Wenn es möglich ist, sollten die Schnittstellen formal (z. B. mit XSD-Dateien) beschrieben werden und die formale Beschreibung sollte als Entwicklungsartefakt von den Entwicklern genutzt werden. Es ist hilfreich, diese Beschreibungen in die Hand des Architekten zu legen, sodass dieser ein systemübergreifendes Verständnis der Schnittstellen hat und darauf einwirken kann, die Schnittstellen in ihre einfachste Form zu bringen.

Fazit

Die hier gemachten Vorschläge haben ausdrücklich nicht das Ziel, das Scrum-Framework komplizierter zu gestalten oder eine zusätzliche Hierarchieebene für den Architekten zu etablieren. Auch sollen klassische Architekturwerkzeuge, von denen ich hier nur einige genannt habe, nicht ersetzt werden. Vielmehr sind die Ausführungen als Präzisierung der Architektenrolle im Kontext des teamorientierten Scrum-Rollenmodells zu verstehen.

Dieser Artikel ist ein Plädoyer für die Rechte und Pflichten des Architekten im Scrum-Prozess, für die Zusammenarbeit im Scrum-Team und gegen die Architektur durch Gremien. Ich hoffe, Sie können einige der Vorschläge mit Gewinn in Ihrem aktuellen Projekt oder bei einem zukünftigen Vorhaben einsetzen und wünsche Ihnen viel Erfolg dabei. ■

[1]) *Von „to mock", etwas vortäuschen.*

Literatur & Links

[arc42] P. Hruschka, G. Starke, arc42, Ressourcen für Softwarearchitekten, siehe: http://arc42.de/template/template/02-constraints.html

[Bec01] K. Beck et. al., Agiles Manifest, 2001, siehe: http://agilemanifesto.org

[Bro95] F.P. Brooks, The Mythical Man Month (überarbeitete Ausgabe), Addison Wesley 1995

[Bro10] F.P. Brooks, The Design of Design, Addison Wesley 2010

[Con68] M.E. Conway, How Do Committees Invent? 1968, siehe: http://www.melconway.com/Home/Conways_Law.html

[Fri10] U. Friedrichsen, Wer braucht einen Architekten? Über Ziele und Aufgaben von Architektur und Architekten, in: OBJEKTspektrum 03/2010

[Pic09] R. Pichler, The Product Vision, 2009, siehe: http://www.scrumalliance.org/articles/115-the-product-vision

[Pre10] L. Prechelt, Vorlesung Softwaretechnik, 2010, siehe: http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2010/31_Architektur.pdf

[Sch04] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Pearson Prentice Hall 2004

[Sie04] J. Siedersleben, Moderne Softwarearchitektur, umsichtig planen, robust bauen mit Quasar (Überarbeitete Ausgabe), dpunkt.verlag 2006

[Zör09] S. Zörner, Historisch gewachsen? - Entscheidungen festhalten, in: Java Magazin 04/2009

[Zör10] S. Zörner, Gretchenfrage 2.0: Was unterscheidet Softwarearchitekten von Entwicklern?, in: Java Magazin 10/2010