Was fehlt Scrum? – ein beispielhafter Lösungsansatz aus der Hochschulpraxis

Jörg Hofstetter, Martin Jud

Hochschule Luzern Technik & Architektur
Technikumstrasse 21, 6048 Horw, Schweiz
joerg.hofstetter@hslu.ch
martin.jud@hslu.ch

Abstract: Der Einsatz eines agilen, iterativ inkrementellen Vorgehensmodells wie Scrum ist heute in der Software-Industrie State of the Art. Wesentliche Aspekte von Softwareprojekten, wie der Umgang mit ändernden Anforderungen oder der Druck auf eine frühe Auslieferung, können damit sehr gut beherrscht werden.

Es gibt aber auch wichtige Aspekte der Software-Erstellung welche in Scrum nur marginal oder gar nicht unterstützt werden. Dies betrifft insbesondere den Umgang mit Architektur-Fragen, komplexen Anforderungen und die Integration in einen Projektmanagement-Gesamtrahmen. Die fehlende Unterstützung durch Scrum für diese Aspekte wird auch von andern Autoren als Mangel empfunden. Sie zeigen aber auch, dass sich eine Unterstützung dieser Aspekte durchaus mit Scrum vereinbaren lässt. Wie dies aussehen kann – ohne die ursprüngliche Kraft und Einfachheit von Scrum zu gefährden – zeigen wir Beispielhaft etwas genauer an einem konkreten für unsere Hochschule entwickelten Vorgehensmodell.

1 Warum Scrum?

1.1 Warum agiles, iteratives Vorgehen in Software Projekten essentiell ist

Ingenieurwissenschaften wie z.B. das Bau- und Maschineningenieurwesen haben über Jahrzehnte hinweg erfolgreiche Vorgehensmodelle geschaffen um höchst komplizierte Probleme zu lösen. In den Anfängen der Informatik orientierte man sich naheliegenderweise an diesen etablierten Modellen, musste aber feststellen, dass eine einfache Übertragung nicht möglich ist.

In den klassischen Ingenieur-Disziplinen kennt man eine klare Trennung zwischen den Bereichen Konzipieren (Entwerfen, Konstruieren) und dem eigentlichen Produzieren (Herstellen, Umsetzen). Neben einer zeitlichen Entkopplung dieser Bereiche, sind es typischerweise auch verschiedene Personen, welche mit unterschiedlichen Skills die entsprechenden Tätigkeiten ausführen. Meist fällt der grössere Teil der Kosten im Bereich der Produktion an.

Die Herstellung von Software unterscheidet sich davon grundsätzlich: Konzipierung und Implementierung werden oft vom gleichen Team aus Personen mit ähnlichen Skills ausgeführt. Die Arbeitsteilung Analytiker / Programmierer hat sich nicht halten können, heute erfolgt die Arbeitsteilung in Software-Projekten eher entlang von Komponenten und Subsystemen als zwischen den Herstellungsphasen. Friedrichsen [Fr10] bezeichnet Konzipieren und Realisieren als Entwurfsphase, die im Gegensatz zu den klassischen Ingenieurdisziplinen relativ teuer ist und grenzt diese gegen die Bauphase – das eigentliche Produzieren – ab, das im Wesentlichen bei der Software ein Kopieren und damit kostengünstig und schnell ist.

Software und damit Software-Projekte umfassen viele Einflussgrössen, welche eng und dynamisch miteinander gekoppelt sind. Eine getroffene Annahme kann das gesamte System stark beeinflussen und zu neuen Randbedingungen und Möglichkeiten führen. Siehe auch „Spezielle Eigenschaften der Software“ in [LL10].

Während es z.B. beim Bau einer Brücke kaum denkbar ist, in einem ersten Schritt eine „normale“ Brücke zu bauen, diese dann in einem weiteren Schritt auf eine 4-spurige Brücke auszubauen, sind solche Dinge in der Software-Erstellung durchaus möglich. Die iterativ- inkrementelle Herangehensweise, d.h. die laufende und parallele Weiterentwicklung von Anforderungen, Architektur/Design und Umsetzung, eröffnet neue Möglichkeiten und Chancen um innovative Softwareprodukte auf dem Markt zu lancieren, mündet aber auch in einer entsprechenden Prozess-Komplexität.

In einem solchen Umfeld bieten klassische, eher lineare Vorgehensweisen kaum Vorteile gegenüber eher empirischen Problemlösungen. Nicht zuletzt weil in der Software-Entwicklung die Kosten für die Realisierung nicht wesentlich höher sind als für die Konzeptionierung, drängt sich ein schrittweises Vorgehen, ein „Herantasten“ auf.

Ein Projekt- und Vorgehensmodell für den Softwarebereich muss aus unserer Sicht zwingend auf diese Eigenheiten eingehen. Es ist sicher kein Zufall, dass iterativ- inkrementelle Vorgehensmodelle vor allem in der Softwareherstellung entwickelt wurden und inzwischen in der industriellen Software-Praxis weit verbreitet sind (siehe dazu auch die Studie [KM12]). Die wachsende Komplexität auch in anderen Bereichen der Technik und die immer besseren Möglichkeiten im Prototypenbau ebnen nun den Weg für den vermehrten Einsatz solcher Vorgehensmodelle auch in anderen Disziplinen.

In der industriellen Software-Praxis verzeichnen wir aktuell einen starken Trend zum Einsatz des Vorgehensmodells Scrum [SAL]. Aus unserer Sicht zeichnen sich Scrum und ähnliche Vorgehensweisen insbesondere durch folgende Erfolgsfaktoren aus:

- Taktung der Arbeit: Durch klar definierte Iterationen ergibt sich im Projekt eine klare Arbeits-Taktung. Die Mitarbeitenden haben klare Ziele vor Augen und diese werden in überschaubaren Intervallen überprüft. Im Idealfall stellt sich nach einigen Iterationen ein produktiver „Arbeits-Flow“ ein.

- Definition of done: Es werden messbare Ziele gesetzt. In der Software-Herstellung kann dies erfahrungsgemäss nur lauffähige, getestete Software sein.

Damit kann der Gefahr von nicht umsetzbaren Konzepten und ewig unfertiger Software begegnet werden.

- Durchgängiger Einbezug der Fachebene in den Entwicklungsprozess: Bei einem agilen Vorgehen werden laufend wichtige Entscheide getroffen. Die Entwickler können und sollen diese nicht alleine fällen. Ein „Produktverantwortlicher" (Product-Owner) muss entsprechend in die Pflicht genommen werden, d.h. die fachliche Ebene wird in die Entwicklung eingebunden.

2 Was fehlt Scrum?

Obwohl Scrum als Vorgehensmodell in Software-Projekten zunehmend und durchaus erfolgreich eingesetzt wird, sei die Frage erlaubt: Ist mit dem Einsatz von Scrum das Thema Vorgehen und Projektmanagement für ein Projekt abschliessend behandelt? - Wir meinen nein! Scrum stellt ein sehr hilfreiches Instrument dar, welches aber (in vielen Projekten) ergänzende Elemente benötigt.

In den nachfolgenden Kapiteln sollen folgende Aspekte beleuchten werden, welche Scrum nur marginal abdeckt:

1. neuartige, grosse Systeme, für die zuerst eine Systemarchitektur entworfen werden muss

2. Komplexe Ausgangslagen, die eine vorgängige Situations- und Anforderungsanalyse erfordern

3. Softwareprojekte im Rahmen technischer Produktentwicklung (HW/SW-CoEntwicklung)

Die mangelnde Unterstützung für diese Aspekte hat wohl damit zu tun, dass gemäss Scrum Ur-Paper von Schwaber [Sc95] Scrum sich explizit auf Software-Wartung beschränkt:

> *Scrum is concerned with the management, enhancement and maintenance of an existing product, (. . .) Scrum is not concerned with new or re-engineered systems development efforts.*

Im aktuellen Scrum Guide [Su11] steht genau das Gegenteil:

> *Scrum wird (. . .) bei der Umsetzung komplexer Produktentwicklungen verwendet.*

Allerdings hat sich methodisch in Scrum nichts verändert, was diesen Wandel im Anspruch unterstützen könnte. In der Software Community wollen viele das Potential nutzen, das unbestritten in Scrum steckt und haben entsprechend nach Lösungen gesucht, um die angesprochenen Problemfelder beherrschen zu können.

2.1 Systemarchitektur und Scrum verbinden

Die Stärken von Scrum kommen dann zum Tragen, wenn für eine bestehende Architektur Anpassungen oder Erweiterungen entwickelt werden. Was aber, wenn in einem Projekt eine ganz neue Architektur mit entsprechenden Risiken entwickelt werden muss? Wir stellen fest, dass das Thema Systemarchitektur in Scrum eher am Rande behandelt wird.

Die Frage, wie Anforderungen technisch umgesetzt werden, also die Software-Architektur bzw. der Systementwurf, erfolgen gemäss Scrum Guide in der zweiten Phase der Sprint Planung. Architektur ist damit eine Tätigkeit, welche pro Sprint durchgeführt wird.

In Projekten, in welchen die Systemarchitektur gesetzt ist z.B. bei Software-Erweiterungen, mag dies ein valables Vorgehen sein. Es gibt aber auch Projekte, in denen eine neue Architektur geschaffen werden muss, deren Komplexität und Risiken den Einsatz erfahrener Software-Architekten und möglicherweise eine vorgängige Erstellung und Machbarkeitsabklärung einer Gesamtarchitektur erfordern.

Das „Scaled Agile Framework“ [SAF], [Le11] von Dean Leffingwell erweitert den Einsatz von Scrum auch für grosse Projekte mit komplexen Architektur- und Anforderungsfragen. Dazu wird eine separate Portfolio-Ebene eingeführt, so können Architektur- und Business-Epics über mehrere Iterationen geplant und verfolgt werden. Dabei bleibt allerdings offen, wie „design spikes“ und „value stories“ priorisiert und in den Sprints umgesetzt werden.

2.2 Anforderungsanalyse und Scrum verbinden

In manchen Projekten braucht es eine vorgängige Anforderungsanalyse, in welcher Anforderungen nicht (nur) als User Stories festgehalten werden. Ist damit der Einsatz von Scrum bei der Umsetzung nicht mehr möglich?

„Ein Projekt agil durchzuführen, bedeutet nicht, ohne vorherige Spezifikation des Gesamtsystems mit der Umsetzung zu beginnen“ [Ec10]. Eckkrammer et. al. stellen der Scrum Projektphase eine Vorprojektphase zur Anforderungsdefinition voran. Darin wird eine Risikoanalyse gemacht und nötigenfalls werden Prototypen gebaut. Das schafft die Basis für die Auftragserteilung und liefert einerseits den initialen Produkt-Backlog, andererseits aber auch eine übergeordnete Releaseplanung, Aufwand- und Kostenschätzung.

Insbesondere bei grösseren Projekten gibt es Anforderungen, welche zu gross sind, um sie in einer User Story zu fassen und in einer Iteration umzusetzen:

> *What happens when a story includes too many unknowns to tell just how big it is? Or what if the story’s requirements are known, but its effort is too huge to complete in a single sprint? We call these stories “epics.” [SAL]*

Epics werden im Product-Backlog festgehalten und während der Spint-Planung in umsetzbare Stories zerlegt. Dabei macht es durchaus Sinn, die Beziehung der abgeleiteten Stories zu den ursprünglichen Epics aufrecht zu erhalten z.B. indem wichtige Anforderungen während einer frühen Konzeptionsphase in einem Konzeptpapier (Anforderungsanalyse) festgehalten werden.

Auch die Firma XING AG ergänzte das agile Vorgehensmodell mit einer übergeordneten Projektführungsschicht (Meta-Ebene) in der die selbstorganisierten Teams als Durchführungsschicht eingebunden wurden. Das Projekt wurde in drei Phasen (Vorbereitungsphase, Phase des verteilten Arbeitens, Konvergenzphase) gegliedert und eine verbindliche Grobplanung (Timebox für das Gesamtprojekt) festgelegt [KM12]. Der XING-Plattform-Relaunch stellte XING vor besondere Herausforderungen, weil die für Enhancement und Maintenance agil aufgestellte Entwicklung nun ein Grossprojekt realisieren musste.

2.3 Interdisziplinarität und Scrum verbinden

Scrum modelliert den Prozess der Software-Erstellung. Oft ist aber die Software-Erstellung nur ein Teil eines Projektes. Sei es, weil im Projekt auch Hardware entwickelt wird oder weil z.B. im Projekt anhand einer Markt-, Kosten- und Machbarkeitsanalyse vorgängig entschieden werden muss, ob das Vorhaben überhaupt umgesetzt wird. Dies bedeutet, dass Scrum in ein Gesamtvorgehen einzubetten ist.

Im Kontext von technischer Produktentwicklung, wo die Softwareentwicklung ein Teil des gesamten Entwicklungsprojektes ist, hat z.B. ABB schon sehr früh aufgezeigt, wie die iterativ-inkrementelle Softwareentwicklung in das firmeninterne Gate Model von ABB integriert werden kann [Wal02], [KR06]. Dabei wird das firmeninterne Gate Model von ABB als Projektführungsschicht der iterativ-inkrementellen Softwareentwicklung als Projektdurchführungsschicht übergeordnet.

Im folgenden Kapitel versuchen wir für studentische Projekte aber auch für anwendungsorientierte Forschungsprojekte im Informatikbereich einen Rahmen zu schaffen, der einerseits von der Einfachheit und Kraft von Scrum profitiert, aber andererseits die oben aufgeführten Aspekte bezüglich Architektur und Anforderungsmanagement trotzdem unterstützt.

3 SoDa - Ein beispielhafter Lösungsansatz, angewendet in der Hochschulpraxis

Das Informatikstudium an der Hochschule Luzern – Technik & Architektur vermittelt eine fundierte Fachausbildung für eine professionelle Tätigkeit als Informatikerin oder Informatiker. Das Curriculum ist modular aufgebaut. Die Informatik-Kernausbildung vermittelt essenzielle Kompetenzen in den Bereichen Software-Entwicklung, Systemtechnik, Modellierung und IT-Projektmanagement.

Weil sich der Bereich Software-Entwicklung über das gesamte Studium erstreckt und von verschiedenen Dozentinnen und Dozenten in Modulen unterrichtet wird, die teils mehr theoretisch ausgerichtet sind, teils mehr Projektcharakter haben, haben wir uns entschlossen, ein Projekt- und Vorgehensmodell anzubieten, das den Studierenden als Klammer über alle Module im Bereich Software-Entwicklung dient.

Das Ergebnis heisst SoDa (SoDa steht für Software Development agile). Wir stellen es nachfolgend kurz vor.

3.1 Anforderungen an ein Projekt- und Vorgehensmodell

Ein Projekt- und Vorgehensmodell soll uns in der Lehre in den oben erwähnten Fachmodulen aber insbesondere auch in interdisziplinären Projektmodulen mit den Abteilungen Elektrotechnik und Maschinenbau dienen. Ausserdem soll es uns auch in der Institutsarbeit, d.h. in der angewandte Forschung und Entwicklung bei der Projektzusammenarbeit mit der Industrie unterstützen.

Die folgenden Anforderungen haben wir vor diesem Hintergrund zusammengestellt:

- Nähe zur Industrie
  das Projekt- und Vorgehensmodell soll strukturell und terminologisch nicht zu stark vom in der Wirtschaft und Industrie praktizierten Vorgehen abweichen.

- Aktualität
  das Projekt- und Vorgehensmodell soll zeitgemäss sein, d.h. es soll dem Stand der Technik entsprechen.

- Anwendbarkeit
  das Projekt- und Vorgehensmodell soll bei den in Frage stehenden Projekten (Forschungsprojekte mit der Industrie, schulinterne interdisziplinäre Projekte und Lernprojekte) sinnvoll anwendbar sein.

- Umsetzbarkeit
  das Projekt- und Vorgehensmodell soll in typischen studentischen Projekten umsetzbar sein (Anzahl und Art der Rollen, zeitliche Aspekte, beurteilbare Artefakte)

In Industrie und Wirtschaft hat Anfang des letzten Jahrzehnts – ausgehend von mittelständischen Unternehmen – schrittweise eine Abkehr von klassischen rigiden Vorgehensmodellen und eine Hinwendung zum agilen Entwickeln eingesetzt [Ju09]. Inzwischen hat sich dieser Trend fortgesetzt, dabei wird offensichtlich, dass unter den agilen Ansätzen Scrum das Rennen macht und sich als de facto Standard – wenn auch nicht immer vollständig umgesetzt – etabliert [KM12].

3.2 Randbedingungen im schulischen Umfeld

Im schulischen Umfeld dauern Projekte oft nur wenige Wochen (z.B. Lernprojekte während dem Semester) oder Monate (Bachelor Diplomarbeiten). Dabei sind typisch ein bis vier Studierende beteiligt. Die von Scrum vorgegebenen Rollen Product-Owner, Scrum Master und Entwickler (vgl. Scrum Guide [Su11]) können daher nur teilweise ausgefüllt werden.

Die absolutistische Machtstellung des Product-Owners gemäss Scrum Guide „Die Entscheidungen des Product-Owners müssen durch die gesamte Organisation respektiert werden, anderenfalls kann der Product-Owner seine Rolle nicht verantwortlich ausfüllen und auch keinen Erfolg haben." lässt sich schlecht mit der Rolle von Studierenden in Lern- oder Wirtschaftsprojekten bzw. Diplomarbeiten in Einklang bringen. Es sei denn der Product-Owner wird als Rolle von Dozierenden bzw. Wirtschaftspartnern wahrgenommen, was diesen nicht unbedingt zuzumuten ist.

Des Weiteren besteht von der Schule der Anspruch, dass die Studierenden sich ernsthaft mit der Architektur der zu entwickelnden Software auseinander setzen und diese, sowie die zu Grunde liegenden Architekturentscheidungen in nachvollziehbarer Form dokumentiert sein müssen. Scrum steht dazu zwar nicht direkt im Widerspruch, marginalisiert aber die Bedeutung der Architektur. Und viele (miss-)verstehen das agile Manifest so, dass „over comprehensive documentation" dazu legitimiert, die Entwurfsdokumentation wegzulassen.

Last but not least sind schulische Projekte in der Regel zeitlich und organisatorisch in einen starr reglementierten Rahmen der Semesterplanung sowie Testat- und Abgabetermine eingebunden. In zwei oder drei, bei Diplomarbeiten vielleicht höchstens fünf Sprint müssen die Ziele erreicht sein und die zu bewertenden Ergebnisse vorliegen. Andernfalls sind Testat- und Abgabebedingungen nicht eingehalten.

3.3 PM-Rahmen als Ausweg aus dem Dilemma

Aus den obigen Gründen kann im realen schulischen Umfeld Scrum nicht „as is", jedenfalls nicht praktisch, d.h. in Lern-, Wirtschafts- und Diplomarbeitsprojekten, eingesetzt werden. Wer also unter den oben genannten Rahmenbedingungen behauptet, „wir machen an unserer Schule Scrum" ist nicht ehrlich und / oder nicht konsequent. An der Abteilung Informatik der Hochschule Luzern – Technik & Architektur, haben wir uns deshalb entschieden, basierend auf Scrum ein eigenes Projekt- und Vorgehensmodell aufzusetzen, das den spezifischen Ansprüchen in unserem Umfeld genügt.

Unseren Überlegungen haben wir das Projektmanagement-Modell von Jenny [Je03] zu Grunde gelegt, welches uns im Hochschul-Alltag gerade in interdisziplinären Diskussionen zu Vorgehensfragen immer wieder klärend zur Seite steht.

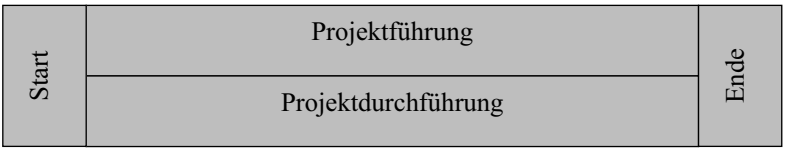

Abbildung 1: Projektmodell nach Jenny (vereinfacht)

Das Modell unterscheidet im Wesentlichen zwei unterschiedliche Ebenen. Einerseits die Projektdurchführung, welche an die jeweilige Disziplin und Projektart angepasst werden muss, und andererseits der Projektführung, welche unabhängig von disziplinären Vorgehensmodellen ist, das Gesamtprojekt umfasst und der Kommunikation und Entscheidungsfindung mit dem Management dient.

Während in der Projektführung Planung und Controlling im Vordergrund stehen, geht es bei der Projektdurchführung um Konzipierung, Realisierung und Einführung. Laut Jenny können in der Projektdurchführung abhängig von den Disziplinen und der Projektart die unterschiedlichsten Vorgehensmodelle zu Einsatz kommen (und bei Projekten mit mehreren Disziplinen auch gemischt werden).

Für unseren Hochschulalltag, aber auch für unsere Projektzusammenarbeit mit der Industrie hat sich das Projektmanagement-Modell von Jenny sehr bewährt. Es schafft einen Rahmen in dem das Projekt abgewickelt wird, dabei verpflichtet sich der Auftragnehmer (Studierende) auf inhaltliche und zeitliche Zielsetzungen, wie sie mit dem Auftraggeber (Lehrpersonen bzw. Wirtschaftspartner) vereinbart wurden – eine Grundvoraussetzung für jegliche Lern- und Wirtschaftsprojekte bzw. Diplomarbeiten.

Die Studierenden stehen damit nach dem Zwei-Ebenen-Modell von Jenny auch in der Projektführungs-Verantwortung. Es gibt eine Projektleiter-Rolle. Das Projekt hat einen definierten zeitlichen Rahmen (Semester-, Testat-, Abgabetermine) völlig losgelöst vom technischen Vorgehen. Der Projektablauf ist konventionell strukturiert, d.h. es gibt einen Projektstart mit Initialisierungsphase und ein Projektende mit einer Einführungsphase.

Auf dieser Basis kann insbesondere auch mit Wirtschaftspartnern, deren Kerngeschäft nicht die Informatik ist – sei dies ein Maschinenbauer, eine Anlagenbau-Firma oder ein Finanzdienstleister – klar und einfach kommuniziert werden.

Abbildung 2: Einordnung des iterativ-inkrementellen Vorgehens (angelehnt an [Je03])

Das Projektmodell von Jenny geht explizit davon aus, dass je nach Projektart unterschiedliche Vorgehensmodelle und Charakteristika der Projektphasen anzuwenden sind. Insbesondere ist es bei Software-Entwicklungsprojekten meist nicht möglich bereits in der Initialisierungsphase die Anforderungen vollständig zu erfassen. Zudem ist die Trennung zwischen Konzeption (SW-Design) und Realisierung (Programmierung) bei der Software-Entwicklung eher künstlich.

In SoDa tritt deshalb an Stelle der konventionellen Konzeptions- und Realisierungsphasen ein iterativ-inkrementellen Vorgehen, das sich stark an Scrum anlehnt. Scrum wird hier also im Sinne eines Vorgehensmodells für einen Teil der Projektphasen innerhalb eines konventionellen Projektrahmens genutzt. Die Projektleiter-Rolle mutiert für diesen zentralen Projektabschnitt zum Scrum-Master und unterstützt als «servant leader» das Team und den Product-Owner.

3.4 Planung: lieber grob richtig als genau falsch

Nach wie vor gilt die unter dem Namen Cone of Uncertainty bekannte Beobachtung von Barry Boehm, der bereits 1981 festgestellt hatte, dass Aufwandschätzungen zu Beginn eines Projektes mit einem Unsicherheitsfaktor von mal vier / durch vier behaftet sind [Bo81]

Mit so viel Unsicherheit (25% – 400%) ganz am Anfang, kann weder der Auftraggeber noch der Auftragnehmer vernünftig leben. Ein zweistufiges Vorgehen hilft, mit dieser Situation zurecht zu kommen:

1. Rahmenplan: Der Projektleiter erarbeitet deshalb einen Rahmenplan, der auf der Ebene des Projekt-(Phasen-)Modells eine grobe Vorstellung des übergeordneten Projektablaufs mit den wichtigsten Meilensteinen und Terminen gibt.

2. Rollende Detail-Planung: Der Product-Owner priorisiert laufend die Anforderungen im Backlog nach Geschäftsnutzen. Gemeinsam mit dem Entwicklungsteam wird der Aufwand für die höchst priorisierten Anforderungen abgeschätzt und die jeweils folgenden zwei Iterationen werden detailliert geplant.

Dieses Vorgehen erlaubt, den Ablauf des Projektes schon früh grob richtig festzuhalten. Die Planungsunsicherheit wird durch die Priorisierung des Backlogs aufgefangen: hoch priorisierte Anforderungen werden garantiert umgesetzt und ausgeliefert.

Die 100% vollständigen Requirements, die 100% perfekte Spezifikation sind erst nach unendlich langer Zeit zu haben – «gut genug» ist das Ziel: eine kurze Initialisierungsphase muss genug Klarheit schaffen, um mit der Umsetzung rasch beginnen zu können. Am Ende der Initialisierungsphase, d.h. an der Schnittstelle zwischen Projektmanagement-Rahmen und iterativ-inkrementeller Entwicklung, steht einerseits der initial aufbereitete und priorisierte Product-Backlog, andererseits eine Release-Planung (denn Projektziele und Abgabetermine stehen in Lern- oder Wirtschaftsprojekten bzw. Diplomarbeiten ja fest). Der Product-Owner verteilt die Backlog Items auf die Sprints, er prüft am Ende jedes Sprints was „done“ ist, gibt das Inkrement frei und ermittelt die verbleibende Arbeit.

Dieses Modell trägt einerseits dem Bedürfnis von Auftraggebern und übergeordneter Organisation nach geordneter Projektdurchführung mit standardisierten Phasen und Meilensteinen Rechnung. Andererseits wird sichergestellt, dass die Software iterativ-inkrementell entwickelt wird und sich agil an den sich möglicherweise ändernden Bedürfnissen der Auftraggeber und an den Möglichkeiten des Entwicklungsteams orientiert.

3.5 Was du schwarz auf weiss besitzt

Bleibt noch die Frage der Bewertbarkeit und Nachvollziehbarkeit der geleisteten Projektarbeit. Ein offensichtlicher Anspruch in Schulprojekten, aber natürlich nicht nur da. David L. Parnas, der Pionier der modernen Softwaretechnik, hat einmal gesagt:

> *At the root of all the problems we have with software lies the failure of software developers to document design decisions in a way that allows those decisions to be reviewed, to guide the implementors, to guide the testers and to guide those who will maintain it in the future. [Pa08]*

SoDa kennt zwei Kategorien von Dokumenten: die Entwurfsdokumente, welche im Parnas’schen Sinn die Software dokumentieren und die Planungsdokumente welche den Entwicklungsprozess unterstützen und dokumentieren.

Planungsdokumente gibt es einerseits auf der Projektmanagement-Ebene andererseits auf der Ebene der Sprint-Planung.

Der SoDa Projektplan lehnt sich an den IEEE Software Project Management Plan an (IEEE Std 1058-1998), allerdings im Sinne des agilen Manifests auf das Notwendigste reduziert. Die Terminplanung beschränkt sich explizit auf den Rahmenplan (siehe oben).

Der SoDa Sprintplan enthält Sprint-Ziel, das Sprint-Backlog, das jeweils zu Beginn eines Sprints vom Product-Owner mit dem Team auf Basis der Rahmenplanung und des Product-Backlogs erarbeitet wird. Das Sprint-Backlog ist eine Liste aller Items, die vom Team im Rahmen des nächsten Sprints erledigt werden müssen, um das Sprintziel zu erreichen. Dazu gehören pro Item deren „Definition of done", eine Aufwandschätzung und falls nötig eine Zerlegung in Personen zugeordnete Tasks.

Typischerweise handelt es sich bei den Backlog-Items um User-Stories (Produkt-Funktionalität). Die Umsetzung einer User-Story umfasst neben dem Codieren und Schreiben der Unittests auch Tätigkeiten wie Requirements verfeinern und dokumentieren, Design-Entscheide ergänzen und dokumentieren, Integrations- und Systemtests planen und durchführen, Entwicklungs- und Testumgebung erweitern und pflegen, Refactoring etc. Daneben erlaubt SoDa auch Backlog-Items, welche nur Teile davon umfassen, wie z.B. die Dokumentation und Diskussion einer Architekturentscheidung.

SoDa Entwurfsdokumente, welche im Parnas'schen Sinn die Software dokumentieren, sind einerseits die Verschriftlichung von Product-Backlog und Sprint-Backlog andererseits die Systemspezifikation und der Testplan. Entsprechend dem iterativ-inkrementellen Vorgehen werden die Dokumente in der Initialisierungsphase erstellt, sind aber naturgemäss am Ende der Phase nicht vollständig und müssen deshalb bei jedem Sprint ergänzt und allenfalls korrigiert werden.

Die SoDa Systemspezifikation ist im weitesten Sinne an die Software Design Description (IEEE Std 1016-1998) angelehnt. Sie umfasst vier Abschnitte. Systemübersicht (1): Übersicht und Begründung des gewählten Lösungsansatzes. Architektur und Design-Entscheide (2): Modell(e) und Sichten, Daten (Mengengerüst / Strukturen), Entwurfsentscheide. Schnittstellen (3): Externe Schnittstellen, wichtige interne Schnittstellen, Benutzerschnittstelle(n). Environment-Anforderungen (4): Anforderungen bezüglich Hardware, Betriebssystem etc.

Ein zentrales Element der Qualitätssicherung in Scrum ist die Definition of done. Dafür erforderliche Integrations- und Systemtestfälle lassen sich oft nicht automatisieren. In SoDa wird deshalb ein Testplan in jedem Sprint mit den neu dazu gekommenen Testfall-Beschreibungen erweitert damit jederzeit reproduzierbare Regressionstests gemacht werden können. Der Testplan ist das Werkzeug des Product-Owners bei der Sprint-Abnahme.

4. Fazit

Agiles, iteratives Vorgehen ist in der Software-Entwicklung essentiell. Als Vorgehensmodell setzt sich Scrum in der Industrie zunehmend durch. Für Neuentwicklungen, in denen Requirements-Engineering, Architektur und Systementwurf ein höheres Gewicht haben, aber auch für Produktentwicklungen mit mehreren Disziplinen deckt Scrum nicht alle projektrelevanten Aspekte zufriedenstellend ab.

In manchen Projekten braucht es daher eine übergeordnete Planungsebene. Verschiedene Modell-Ansätze ordnen Scrum in ein mehrschichtiges Modell ein, es schafft Rollen und Ansatzpunkte um den in Scrum fehlenden Aspekten gerecht zu werden. Dies darf aber keineswegs dazu führen, dass Scrum in seiner Einfachheit und Kraft eingeschränkt wird. Für unser Hochschulumfeld haben wir diesen Weg mit SoDa exemplarisch umgesetzt.

Bei der Beurteilung der ersten Erfahrungen mit SoDa im Unterricht vergleichen wir das neue Modell mit dem ebenfalls an unsere Schule geschaffenen und über mehrere Jahre eingesetzten Vorgänger „HTAgil“[1], welcher sich stark am Vorgehensmodell RUP (Rational Unified Process) orientierte.

Wir können folgendes, erstes Fazit ziehen:

Auf Ebene der einzelnen Sprints (Mikroplanung) konnte eine klare Verbesserung des Vorgehens der einzelnen studentischen Gruppen erreicht werden. Die Arbeit ist besser getaktet, die Definition of done wird klarer formuliert weil adäquatere Strukturen die Aufgabe für die Studierenden greifbarer machen. Zudem ist es einfacher geworden klare Sprintziele einzufordern.

Insbesondere bei Projektbeginn macht es Sinn auch Architektur- und Infrastruktur-Fragen als Stories zu formulieren (Sichtbarkeit/Transparenz), die ebenfalls auf Sprints heruntergebrochen werden müssen. Um einen Rückfall in eine technisch getriebene Entwicklung zu verhindern, unterscheiden wir klar zwischen User-Stories und z.B. Qualitäts-&Architektur-Stories („design spikes“ und „value stories“ nach Leffingwell [Le11]). Hier werden wir weiter experimentieren, im Bewusstsein eines gewissen Konfliktes mit der Rolle des Product-Owners.

Der gleichzeitige Einsatz von Dokumenten (in welchem z.B. Kontext-Diagramm, Use-Cases und die Architektur beschrieben werden) und Stories hat sich grundsätzlich bewährt, stellt aber einen dauernden (und wohl grundsätzlichen) Drahtseilakt dar.

Die Planung auf zwei Ebenen (Projektführung und Projektdurchführung) ist für die Studierende herausfordern, sie sind tendenziell in der Projektdurchführung verhaftet. Es ist an uns Coaches, die Ebene der Projektführung aktiv einzufordern.

Insgesamt sind wir auf gutem Weg und überzeugt, dass sich den kommenden Semestern einige noch offene Punkte klären werden.

[1] https://wiki.enterpriselab.ch/edu/publication:htagil:htagil

Literaturverzeichnis

[Bo81] Boehm, B.: Software Engineering Economics. Prentice-Hall, 1981.

[Ec10] Eckkrammer, T.; Eckkrammer, F.; Gollner, H.: Agiles IT Projektmanagement im Überblick. In (Tiemeyer, E. Hrsg.): Handbuch IT-Projektmanagement. Hanser, 2010.

[Fr10] Friedrichsen, U.: Agil oder ingenieurmässig? Business Technology Heft Nr. 2.2010, 2010.

[Je03] Projektmanagement – Das Wissen für eine erfolgreiche Karriere, ISBN 978-3-7281-3218-2, vdf Hochschulverlag, Zürich, 2003.

[Ju09] Jud, M.: Kosten und Nutzen von Vorgehensmodellen – eine vergleichende Studie, ObjektSpektrum 1/2009, siehe: http://www.sigs.de/publications/os/2009/01/jud_OS_01_09.pdf

[KM12] Kaiser, T. ; Menden, F.: Ein Modell zur agilen Realisierung von Grossprojekten, Objekt Spektrum 2/2012. Siehe: http://www.sigs.de/publications/Jahres_PDFs/Gesamtheft_OS_02_12_PPuH.pdf

[KM12] Kropp, M.; Meier, A.: Swiss Agile Study 2012 - Schweizer Informatik wird agil. Swiss IT Magazin 12/2012.

[KR06] Karlström, D.; Runeson, P.: Integrating agile software development into stage-gate managed product development. Empir Software Eng (2006) 11: 203–225, Springer Science + Business Media, 2006.

[Le11] Leffingwell, D.: Agile Software Requirements - Lean Requirements Prac-tices for Teams, Programs, and the Enterprise, Addison-Wesley , 2011

[LL10] Ludewig, J.; Lichter, H.: Software Engineering – Grundlagen, Menschen, Prozesse, Techniken. dpunkt, 2010.

[Pa08] Parnas, D. L.: Disciplined Quality Software Construction, Vorlesung an der Universtà de la Svizzera Italina USI, 2008.

[SAF] Scaled Agile Framework, siehe: http://scaledagileframework.com/, Zugriff: 23.5.2013

[SAL] Scrumalliance: Was ist Scrum? http://www.scrumalliance.org/pages/what_is_scrum, Zugriff: 23.5.13

[Sc95] Schwaber, K.: Business Object Design and Implementation Workshop, OOPSLA '95 Position Paper, siehe: http://www.jeffsutherland.org/oopsla/schwaber.html

[Su11] Sutherland, J.; Schwaber, K.; Scrum Guide – Der gültige Leitfaden für Scrum, Scrum.org, Oktober 2011, siehe: http://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20DE.pdf

[Wal02] Wallin, C.; Ekdahl, F.; Larsson, S.: ABB, Integrating Business and Software Development Models, IEEE SOFTWARE Nov/Dec 2002.