# Agile Softwareentwicklung – Erfahrungsbericht eines Oberstufenprojekts im Wahlpflichtunterricht

Peter Brichzin[1]

**Abstract:** Der Praxisbericht zeigt wie die konkreten Methoden Pair Programming, User Stories, Tasks, Task Board und Standup Meeting in einem Softwareentwicklungsprojekt der Oberstufe umgesetzt wurden, welchen Beitrag sie zum Projekterfolg geleistet haben und welche Erfahrungen für das Lehren daraus gewonnen wurden. Letzteres beinhaltet auch einen Blick auf die didaktische Reduktion der Methoden, denn im Vergleich zu agiler Prozesssteuerung aus dem Lehrbuch wurde bewusst auf Methoden wie Schätzen und typische Rollen wie den Scrum Master verzichtet. Trotz anspruchsvoller Projektthemen überzeugten die Ergebnisse der Schülerinnen und Schüler im Vergleich zu den nicht agil erarbeiteten Projektergebnissen der gleichen Zielgruppe aus den Vorjahren. Wesentlicher Schlüssel zum Erfolg war einerseits das Prototyping und andererseits die positive Unterstützung kollaborativen Arbeitens durch die oben genannten agilen Methoden.

**Keywords:** agil, Softwareentwicklung, Projekt, Oberstufe, Erfahrungsbericht, kollaboratives Arbeiten

## 1 Agile Methoden lösen das Wasserfallmodell ab

Eines der ältesten Vorgehensmodelle in der Softwareentwicklung ist das Wasserfallmodell, welches sequentiell die Phasen Anforderungen, Entwurf, Implementation, Überprüfung und Wartung durchläuft. Obwohl es im Laufe der Zeit z. B. in Form vom erweiterten Wasserfallmodell und Spiralmodell weiterentwickelt wurde, hat es in der IT-Branche heute nur noch eine geringe Bedeutung. In den letzten Jahren haben immer mehr Unternehmen zu agilen Methoden gewechselt. Diese verringern den bürokratischen Aufwand, unterstützen eine kontinuierlichen Interaktion mit dem Kunden sowie eine iterative Softwareentwicklung mit hoher Flexibilität bei der Festlegung der nächsten Schritte (vgl agiles Manifest agile Manifest [Be01]). Bei einer im Herbst 2012 von VersionOne durchgeführten Umfrage, gaben 84% der Befragten an, dass in Ihren Unternehmen agile Prozesse eingesetzt werden [Ve12, S.6].

Die Ursachen hierfür sind vielfältig, drei seien hier herausgegriffen:
Bei Projekten mit einer Dauer von mehr als einem halben Jahr, ergeben sich in der Regel durch neue Entwicklungen – auf dem Markt, im Unternehmen, bei angebundenen Softwaresystemen usw. – auch neue Anforderungen an die zu erstellende Software. Agile Methoden sind im Gegensatz zum Wasserfallmodell interaktiv und flexibel in allen Phasen, auf Veränderungen kann schnell reagiert werden.

[1] QAware GmbH, Aschauer Str. 32, 81549 München, peter.brichzin@qaware.de und
LMU München, Institut für Informatik, Oettingenstr. 67, 80538 München, brichzin@tcs.ifi.lmu.de

Findet die Kommunikation zwischen Kunden und Softwaredienstleister (hauptsächlich) nur in der Anforderungserhebung statt, so besteht die Gefahr bei Missverständnissen ein Produkt zu entwickeln, das nur teilweise den Vorstellungen des Kunden entspricht, Änderungen sind in einer späten Phase sehr teuer. Bei einer agilen Entwicklung erhält der Kunde in regelmäßigen Abständen (4 - 12 Wochen) einen Prototypen. Dadurch kann er eigene Erwartungen mit dem Produkt vergleichen und Änderungen hinsichtlich der Anforderungen oder der Priorisierung einsteuern. Nicht zuletzt erhöht sich bei einer agilen Softwareentwicklung die Motivation und Produktivität im Team, da diesem mehr Eigenverantwortung und Selbstorganisation bei Planung und Umsetzung zugestanden wird.

Im Anschluss an die INFOS 2013 hat sich ein bundesweiter Arbeitskreis formiert, der den Gewinn agiler Methoden im Informatikunterricht diskutiert und Scrum Methoden zu Methodenbausteinen für den Schulunterricht didaktisch reduziert hat. Zwei Jahre später zeigen verschiedene Praxisberichte durchweg positive Erfahrungen. Dieser Artikel hier beschreibt den Erfolg agiler Methoden in einem Informatikprojekt in Jahrgangsstufe 11 an einem bayerischen Gymnasium.

## 2 Rahmenbedingungen

Im Lehrplan der Jahrgangsstufe 11 ist ein Projekt zur Softwaretechnik mit ca. 26 Unterrichtsstunden verankert. Die Schülerinnen und Schüler bringen als Vorwissen einerseits aus Jahrgangsstufe 10 Grundlagen zur Objektorientierten Modellierung und Programmierung mit, andererseits aus Jahrgangsstufe 11 praktische und theoretische Kenntnisse zu den Datenstrukturen Liste, Baum und Graph [LP09]. Programmierung grafischer Benutzeroberflächen, Datenbankanbindung, das Entwurfsmuster MVC und vertiefte Methoden der Projektorganisation sind durch den Unterricht vor dem Projekt noch nicht bekannt, werden aber im Lehrplan gefordert. Dadurch reduziert sich, bei drei Stunden Unterricht pro Woche, die Projektarbeitszeit auf 6 – 7 Wochen.

Als Programmiersprache wurde wie auch im Unterricht Java verwendet. Als Entwicklungsumgebung wurde im Unterricht BlueJ verwendet. Im Projekt verwendete ein Teil der Schüler Eclipse bzw. NetBeans.

Der Kurs bestand aus 21 Schülerinnen und Schülern. Die Leistungsstärke des Kurses war sehr inhomogen. Einige Schüler brachten bereits Erfahrungen z. B. in der Programmierung graphischer Benutzeroberflächen und Verwendung professionellerer Entwicklungsumgebungen ein, andere hatten Schwierigkeiten auch kleine Teilaufgaben innerhalb des Teams selbstständig zu lösen.

# 3 Agile Methoden in der Softwareentwicklung

## 3.1 Agiles Manifest

Es gibt eine Vielzahl unterschiedlicher Vorgehensweisen Software agil zu entwickeln, beispielsweise Scrum, KanBan, Extreme Programming. Da die Zielsetzung des Artikels der Einsatz in der Schule ist, sei hinsichtlich des Einsatzes agiler Methoden in Unternehmen auf Literatur wie z. B. [Pi11][An07] bzw. [HeRoLi05] verwiesen. Gemeinsam ist allen Vorgehensweisen das agile Manifest [Be01] mit vier Werten und 12 Prinzipien. Von den Prinzipien sind folgende für die Schule besonders relevant:

- „Lieferung von funktionierender Software in regelmäßigen, bevorzugt kurzen Zeitspannen“
- „Bereitstellung des Umfeldes und der Unterstützung, welche von motivierten Individuen für die Aufgabenerfüllung benötigt wird“
- „Informationsübertragung nach Möglichkeit im Gespräch von Angesicht zu Angesicht“
- „Als wichtigstes Fortschrittsmaß gilt die Funktionsfähigkeit der Software“
- „Einfachheit ist essenziell (KISS-Prinzip)“
- „Selbstorganisation der Teams bei Planung und Umsetzung“

## 3.2 Methodenauswahl

Die für das Oberstufenprojekt ausgewählte Vorgehensweise orientiert sich an Scrum, jedoch wurde die Anzahl der Methoden, Rollen und Ereignisse deutlich reduziert und adaptiert. Folgende Aspekte der Projektorganisation wurden verwendet:

- **Prototyping – iterativ inkrementelle Softwareentwicklung:**
  Regelmäßiger Meilenstein (iterativer Prozess) ist ein lauffähiger und getesteter Prototyp, der mehr Funktionalitäten bietet, als der vorhergehende Prototyp (inkrementell). Jedes Inkrement umfasst dabei alle Schichten (siehe Abb. 1) Die Schüler sehen dadurch nicht nur schrittweise ihre Software wachsen, sondern die Produktausrichtung und Reihenfolge der umzusetzenden Funktionalitäten ist flexibel. Weiterhin werden ungenaue Schnittstellenabsprachen zwischen Teilgruppen des Teams frühzeitig erkannt und können korrigiert werden.

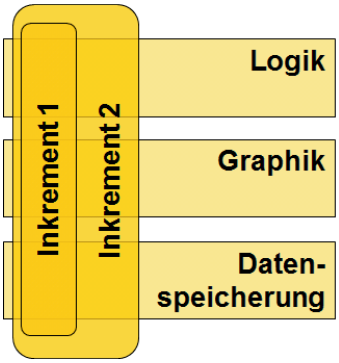


Abb. 1: Inkrementelle Softwareentwicklung über mehrere Schichten

- **Sprint:**
  Das Zeitintervall, in dem ein Prototyp erstellt wird, wird Sprint genannt. Der Sprint beginnt mit einer Planung, hat eine Implementierungsphase und schließt mit Tests ab. In dem Oberstufenprojekt wurde als Zeitintervall von 2 Wochen (6 Schulstunden) gewählt.

- **User Story:**
  Jede User Story beschreibt eine Anforderung aus der Sicht des Endanwenders, d.h. eine Funktionalität der Software. User Stories sind aufgeteilt in Titel und Beschreibung, so kurz formuliert, dass sie auf Karteikarte passen. Sie sind Ergebnis einer Teambesprechung, priorisiert und testbar. Bei der Formulierung einer User Story sind (informatische) Fachausdrücke und Implementierungsdetails verboten.

- **Modeling Story:**
  Für die Schülerinnen und Schüler ist einerseits die Motivation für eine lange Planungsphase gering, andererseits ein vollständigen Systementwurf wie im Wasserfallmodell zu anspruchsvoll. Eine Planung auf Ebene von User Stories (und Tasks) ist eine akzeptierte Grobplanung und kommt den Schülern entgegen, frühzeitig mit ersten Implementierungsschritten zu beginnen. Diese ersten Schritte sind im Lernprozess sehr wichtig, weil die Schüler selbst ein Gefühl erhalten, an welchen Stellen Schwierigkeiten auftreten bzw. ein schnelles Voranschreiten möglich ist (und nicht der Lehrer einen Großteil der konzeptionellen Arbeit abnimmt).
  Jedoch ist bei zunehmender Prototyp-Größe die Zahl der Klassen nicht mehr so einfach zu überblicken bzw. es kann notwendig sein, dass eine sehr zentrale Klasse von mehreren Paaren verändert werden muss. Es entsteht im Entwicklungsverlauf ein Bedarf, einen Überblick über die Klassen, deren Schnittstellen und Beziehungen zu bekommen. Als besondere User Story – Modelling Story - ist es an diesem Punkt sinnvoll, ein Klassendiagramm zu erstellen, welches die Grundlage für weitere Überlegungen und Besprechungen ist.

In der Regel muss der Lehrer den Anstoß dazu geben und die Modeling Story (in einem Standup-Meeting) einsteuern.

- **Task:**
  Tasks sind elementare Aufgaben aus der Sicht des Entwicklers, die durch das Unterteilung einer User Story entstehen Tasks sind aufgeteilt in Titel und Beschreibung, so kurz formuliert, dass sie auf ein „Post it“ passen. Sie sind Ergebnis einer Teambesprechung, testbar und in der Bearbeitung einem Paar zugeordnet.

- **Taskboard:**
  Die gesamte Planung erfolgt über ein im Computerraum aufgehängtes Taskboard (s. Abb. 2). Es dokumentiert über die Spalten „to do“, „in progress“, „in test“ und „done“ zu Erledigendes und bereits Abgeschlossenes, sowie wer aktuell an welchen Arbeitspaketen arbeitet. Auch ist das Sprintende als nächster Meilenstein deutlich vermerkt. Bei den Arbeitspaketen, deren Reihenfolge durch die Priorisierung bestimmt wird, gibt es zwei unterschiedliche Granularitäten: User Stories und Tasks. Niedrig priorisierte müssen nicht von Anfang an als Tasks ausgearbeitet werden.
  Als Taskboards wurden tragbare Tafelelemente bzw. Styroporplatten verwendet. Letztere können schnell auf- und wieder abgebaut werden. Dies ist hilfreich, da der Computerraum von vielen Klassen verwendet wird.

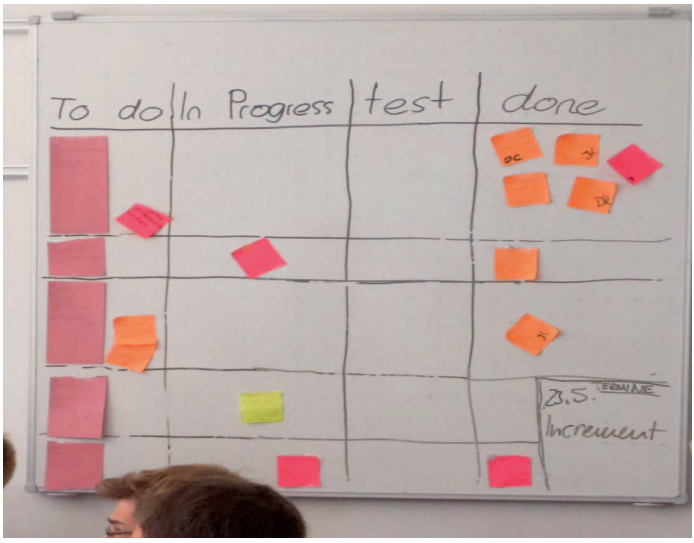


Abb. 2: Taskboard als zentrale Planungsübersicht

- **Programmieren im Pair:**
  Entwickelt wird paarweise, wobei folgende zwei Rollen mindestens einmal pro Unterrichtsstunde gewechselt werden: Der Driver verwendet Tastatur und Maus

und verfasst den Quelltext. Entscheidungen und Absichten werden dem Partner mündlich mitgeteilt. Der Navigator hinterfragt Quelltext, spricht mögliche Fehler an, und sucht elegantere Lösungen. Er hat die Aufgabe das große Ganze im Auge zu behalten.

- **KISS-Prinzip:**
  Die Zeit zum Implementieren ist selten ausreichend. Deshalb ist eine geringe Anzahl an Funktionalitäten (pro Iterationsschritt) und Einfachheit (der Implementierung) sehr wichtig, um sich nicht zu verzetteln: Keep It Small and Simple.

- **Standup-Meeting:**
  Jede Woche findet eine Besprechung im Stehen vor dem Taskboard statt. Dazu äußert sich jedes Teammitglied bzw. jedes Paar knapp und präzise welche Aufgaben es bearbeitet hat. Gegebenenfalls gibt es kurze Hinweise zum Lösungsweg (insbesondere wenn Schnittstellen betroffen sind) und zu aufgetretenen Problemen. Weiterhin nennt jeder den Task, den er als nächstes bearbeiten wird. Die maximale Dauer des Standup-Meetings ist 10 Minuten.
  Diese Besprechung ist wichtig für die Transparenz: Jeder erhält einen Überblick über die aktuellen Aktivitäten, bei der Kommunikation von Problemen können Hilfestellungen gegeben werden. Die Transparenz betrifft in erster Linie das Team. Aber auch für den Lehrer ist das Standup-Meeting ideal, um einen Überblick über den Projektstand zu erhalten.
  Das Standup-Meeting am Ende eines Sprints wird erweitert um die Planung des nächsten Prototyps. Dazu gehört gegebenenfalls eine neue Priorisierung der User Stories, Ergänzung von Tasks und die Aufteilung der Tasks an die Paare.
  Hinweis: Bei den Profis findet dieses Meeting täglich statt (Daily), in der Schule müsste man es Weekly nennen.

### 3.3 Nicht verwendete (Scrum) Methodenbausteine:

Da die Anzahl der Bearbeitertage für die Softwareerstellung in der Schule deutlich niedriger ist als in einem Unternehmen, muss der Aufwand für die Projektorganisation dort auch deutlich niedriger sein, um ein angemessenes Verhältnis zwischen Organisation und Implementierung zu erhalten. Folgende Standardelemente aus Scrum wurden aus diesem Grunde nicht eingeführt:

- **Schätzen - Planning Poker - Burn Down Chart:**
  Das Schätzen von Arbeitsaufwänden setzt Erfahrung voraus und ist für Schülerinnen und Schüler sehr schwer zu bewältigen. Die ist ein Grund, warum auf diese Methoden verzichtet wurde. Ein anderer ist, dass bei einer guten Aufteilung der User Stories in Tasks, für die Schüler "ein Task" sehr bald eine "greifbare" Zeiteinheit wird. Greifbar auch deshalb, weil nach Abarbeitung des Tasks, das entsprechende Post It auf dem Taskboard händisch verschoben wird.

- **Rollen Product Owner und Scrum Master:**
  Neben dem Team sind bei Unternehmensprojekten der Product Owner und der Scrum Master zwei weitere wichtige Rollen. Bei den vergleichsweise kleinen Projekten in der Schule, kann auf die letzten beiden genannten Rollen verzichtet werden. Da die Schüler sich i. A. selbst die Projektidee ausgedacht haben, übernehmen Sie indirekt Aufgaben des Prodct Owners. Sie führen Entscheidungen in der Gruppendiskussion herbei. Eine der Hauptaufgaben des Scrum Masters, (organisatorische) Hindernisse aus dem Weg zu räumen, sollte der Lehrer übernehmen. Er erfährt darüber im Daily.

# 4 Projekteinstieg und -durchführung

## 4.1 Einführung in Agile Softwareentwicklung

In Form eines Lehrervortrags erhielten die Schülerinnen und Schüler eine Einführung in die in Kapitel 3.2 beschriebenen agilen Methoden. Es war in der Projektdurchführung Pflicht diese Methoden zu einzusetzen.

Um das Prototyping begreifbar zu machen, wurde eine Marshmallow Challenge durchgeführt [Wu15]. Aufgabe dabei ist es, mit wenig Material (20 Spagetti, 0,5 m Schnur, 0,5 m Klebeband und 1 Marshmallow) in einer fest vorgegebenen Zeit von 15 Minuten ein möglichst hohes frei stehendes Gebäude zu bauen, dessen höchster Punkt der Marshmallow ist. Durch statistische Auswertungen dieses Wettbewerbs wurde belegt, dass der Erfolg dann größer ist, wenn in der Bauphase der Marshmallow bereits frühzeitig integriert wird, um die Stabilität zu überprüfen. Ausgehend von diesem Prototypen kann dann das Gebäude zum nächsten Prototypen mit einer größeren Höhe weiterentwickelt werden usw. Probleme haben die Gruppen, die erst in letzter Minute den Marshmallow positionieren, denn bei mangelnder Stabilität bleibt keine Zeit für Korrekturen, so dass als Endergebnis die Gebäudehöhe 0cm beträgt.

## 4.2 Themenwahl und Gruppeneinteilung

Vor Projektbeginn hatte jede Schülerin und jeder Schüler den Arbeitsauftrag, über einen Forumseintrag in der Lernplattform des Kurses ein Thema für das Projekt vorzuschlagen. Der Themenvorschlag musste eine Erläuterung in wenigen Sätzen bzw. eine Auflistung von Features enthalten. Die Vorschläge wurden vom Lehrer gruppiert, kommentiert und zur Abstimmung in den Kurs getragen. Die Vorschläge im Einzelnen waren:

- **Spiele:**
  Mario Spiel, Brettspiel TAC, Flappy Stein, Jump ‚n' Run, Black Jack, Poker,

Schafkopf, Space Invaders, Asteroids, Stein, Schere und Papier und Rubiks Cube Löser

- **Anwendungssoftware (ohne Spiele):**
  individualisierte Anzeige des Vertretungsplanes auf dem Handy/Computer, Terminverwaltungsprogramm privat bzw. schulisch (letzteres z. B. mit Klausurenplan, Hausaufgabenliste, Notizen und Noten), Kassensystem für Pausenverkauf, Mediensammlung, Datenverwaltungssoftware (z. B. für schulische Inhalte, wie Hefteinträge und Übungsaufgaben), Nachhilfevermittlung, Bibliothekssoftware für die Lehrmittelbibliothek (zusätzlicher Vorschlag vom Lehrer)

Der Lehrer stellte die Vorschläge vor, kommentierte teilweise (z.B. Nachhilfevermittlung ist nur sinnvoll als Webservice, welcher aber erst im Lehrplan der Jahrgangsstufe 12 Thema ist). Bei der anschließenden Abstimmung gab es in etwa drei gleich große Schülergruppen, mit den Themenwünschen Vertretungsplan (8), Spiel (6) und Bibliothekssoftware (7). Entsprechend der Wünsche wurden drei Teams gebildet. Vorteil dieser Vorgehensweise war, dass für die Themenwahl und Gruppeneinteilung stark auf die Schülerwünsche eingegangen wurde, jedoch nur 20 Minuten Unterrichtszeit benötigt wurden.

### 4.3 Projektdurchführung und -ergebnis

Die Durchführung erfolgte entsprechend der methodischen Vorgaben aus Kapitel 3.2. An das für die Schülerinnen und Schüler neue Pair Programming musste in der Anfangsphase mehrfach erinnert werden, spielte sich aber dann ein. Die interne Organisation konnte jedes Team frei gestalten. Hier ergab sich nicht die Notwendigkeit von Strukturen, denn sowohl die Aufgabenver- und die Paareinteilung erfolgte (über das Taskboard) problemlos und leistungsdifferenziert als auch die Gesprächsleitung in den Weeklies (und damit eine gewisse Leitungsfunktion) übernahm immer jemand initiativ. Jeder dokumentierte seinen Beitrag zum Projekt als knappe Einträge in einem Projekttagebuch.

Nach sechs Wochen wurde der dritte Prototyp zusammen mit dem Taskboard als Dokumentation im Plenum vorgestellt. Die Ergebnisse und die Selbstorganisation waren sehr unterschiedlich. Eine Gruppe hatte nur ein ausreichendes Projektergebnis, scheiterte an mangelhafter Kommunikation, schlechter Organisation (Absenzen von implementierungsstarken Schülern minderte nicht nur die „Manpower“ sondern blockierte auch das gesamte Team wegen fehlender Quelltexte) und nicht realistischer Selbsteinschätzung. Entsprechende mehrfache Hinweise seitens des Lehrers hinsichtlich Fehlentwicklungen, führten nicht wirklich zur Verbesserung. Z. B. blieb ein zusätzlich abzugebender detaillierten Arbeitsplans ein theoretisches Konstrukt und führte nicht wie erhofft zu realistische Umsetzungen. Die beiden anderen Gruppen hatten hervorragende Endergebnisse. Eine davon ausschließlich selbstorganisiert, getrieben von einem hohen Engagement der Mehrzahl der Mitglieder. Das Team glänzte z.B. durch selbst initiierte

Besprechungen, in denen Wissenstransfer von einzelnen Paaren zum Team betrieben wurde. Die andere Gruppe hat Höhen und Tiefen durchlaufen, jedoch Hilfestellungen vom Lehrer gewinnbringend aufgenommen und umgesetzt. Dazu gehörte insbesondere eine Modeling Story nach dem ersten Prototypen, die Klarheit in der Struktur und der weiteren Vorgehensweise sorgte. Ein Motivationsschub gaben hier positive Rückmeldungen des schulinternen Abnehmers der Software.

Ein wichtiger Beitrag für den Fortschritt bei den durchaus komplexen Themenstellungen sind „Hausaufgabenbeiträge". Auch wenn Hausaufgaben wie im „nicht projektorientierten" Unterricht erwartet werden dürfen, fällt das Engagement, ebenfalls wie im „normalen" Unterricht, sehr unterschiedlich aus.

Aus Lehrersicht war die Betreuung dreier unterschiedlicher Themen inhaltlich und organisatorisch anspruchsvoll. Organisatorisch wurde das Weekly zeitversetzt abgehalten, so dass der Lehrer bei allen Gruppen teilnehmen konnte. Nur so ist es möglich einen Überblick zu haben und passend unterstützende Impulse einbringen zu können. Eine inhaltliche Betreuung fand nicht auf Detailebene statt, sondern auf abstrakteren Niveau bzw. mit dem Hinweis auf schülergerechte Quellen, z.B. eine Datenbankanbindung in [Br09 s. 169ff].

## 5 Erfahrungen und Ausblick

Das Oberstufenprojekt wurde bereits das vierte Mal durchgeführt, jedoch erstmals agil. Der Erfolg beim agilen Vorgehen war deutlich besser, einfach messbar an der entwickelten Software, die bisher nicht über eine Alpha-Version hinaus ging, in dem dargestellten Schuljahr jedoch in zwei Gruppen ein Release Niveau erreichte. Im Falle der Schulbibliothekssoftware wurde diese unmittelbar nach Projektende von der Lehrmittelbücherverwaltung eingesetzt. Zentrale Gründe für den Erfolg sind einerseits das Prototyping und andererseits eine Unterstützung des kollaborativen Arbeitens. Das Prototyping reduzierte deutlich Schnittstellenprobleme, die in den letzten Jahren zu hohen Reibungsverlusten geführt hatten und auf Grund begrenzter Zeit nicht mehr aufgefangen werden konnten. Alle anderen agilen Methoden schaffen ideale Bedingungen für ein konstruktivistisches Lernen: User Stories und Tasks sind bei geringem organisatorischen Aufwand eine angemessene Unterstützung der inhaltlichen Planung und Prozessorganisation, um mit selbstbestimmter flexibler Zielsetzung etwas Neues zu schaffen. Das Taskboard sorgt für Transparenz (u.a. auch über den Fortschritt – ein wichtiger Motivationsaspekt). Und die Eigenverantwortung im Team, unterstützt durch das Weekly fördert die Gruppendynamik und das Engagement.

Folgende Optimierungen könnten das agile Vorgehen noch gewinnbringender machen: Einzelne Techniken wie Pair Programming, User Stories/Tasks können schon im Vorfeld in den Unterricht erlernt und angewandt werden. Beim Pair Programming würde ein Wechsel der Partner noch zu mehr Kompetenztransfer führen. Jedoch steht diesem Vorteil ein höherer Zeitaufwand hinsichtlich der Einarbeitung in anderen Bereichen

gegenüber. Eine Unterstützung des kollaborativen Arbeitens durch Versionskontrollsysteme (vgl [BrRa15]) würde den Arbeitsaufwand beim Zusammenführen von Quelltext und andere Reibungsverluste deutlich reduzieren.

Wie das Ergebnis in Kapitel 4.3 zeigt, ist das agile Vorgehen auch kein Garant für Gelingen. (Zu) Hohe Ansprüche der Schüler an Ihre Software stehen in Widerspruch zu der (vielleicht üppig klingenden, aber) knappen Projektdauer von 7 Wochen. Drei Prototypen ist das Minimum um den agilen Prozess erfahrbar zu machen: Der erste Prototyp scheitert meist wegen der Schnittstellen, der zweite zeigt einen gangbaren Weg und mit jedem weiteren Prototypen werden dann zügig Features gebaut. Ein bis zwei Prototypen mehr würden den Schülern mehr Raum für einen Irrweg im Lernprozess sowie allen Beteiligten mehr Zufriedenheit und damit Motivation geben.

Insgesamt sind agile Methoden ideal zur Unterstützung kollaborativen Arbeitens in Projekten geeignet und werden deshalb hoffentlich in Zukunft häufiger in der Schule (und nicht nur im Informatikunterricht) eingesetzt.

## Literaturverzeichnis

[An11] Anderson, D: Kanban: Evolutionäres Change Management für IT-Organisationen, dpunkt.verlag, Heidelberg, 2011.

[Be01] Beck, K. et al.: Manifesto for Agile Software Development, http://agilemanifesto.org/iso/de/ - (zuletzt geprüft am 27.4.2015).

[Br09] Brichzin, P.; Freiberger, U.; Reinold, K.; Wiedemann, A.: Informatik Oberstufe 1, Objektorientierte Modellierung. Oldenbourg Verlag, München, 2009.

[BR15] Brichzin, P.; Rau, T: Repositories zur Unterstützung von kollaborativen Arbeiten in Softwareprojekten, Tagungsband INFOS 2015, Darmstadt, 2015.

[HRL05] Wolf, H.; Roock, S.; Lippert, M: eXtreme Programming: Eine Einführung mit Empfehlungen und Erfahrungen aus der Praxis, dpunkt.verlag, Heidelberg, 2005.

[LP09] Lehrplan Informatik für das Gymnasium in Bayern unter http://www.isb-gym8-lehrplan.de/contentserv/3.1.neu/g8.de/index.php?StoryID=26193&PHPSESSID=e0e818ca1f0e9ec5aca6792e060a29d9 (zuletzt geprüft am 31.01.15).

[Pi07] Pichler, R.: Scrum - Agiles Projektmanagement erfolgreich einsetzen, dpunkt.verlag, Heidelberg, 2007.

[Ve12] Version One, 7th Annual State of Agile Development Survey unter http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf (zuletzt geprüft am 17.1.15) .

[Wu15] Wujec, T.: The Marshmallow Challenge, http://www.marshmallowchallenge.com (zuletzt geprüft am 27.4.2015).