Agile Softwareentwicklungsprojekte in der Schule

Praxisberichte

Wissenschaftliche Artikel

Link zu den deutschsprachigen Beiträgen: http://www.infos15.de/lni.html

Link zu diesem Dokument: https://www.dropbox.com/s/0m24pg62cgocfcu/AgileILTB.pdf?dl=0

Kontakt

Petra Kastl
Friedrich-Alexander-Universität Erlangen-Nürnberg
petra.kastl@fau.de

Agile Softwareentwicklung im Informatikunterricht – Ein Best-Practice-Beispiel am Spiel „Pengu“

Petra Kastl, Silva März und Ralf Romeike[1]

Abstract: Im Beitrag wird der Einsatz agiler Methoden im Informatikunterricht an einem konkreten Beispiel illustriert. Am Spiel „Pengu“, welches mit der Programmierumgebung Greenfoot implementiert werden soll, werden agile Praktiken wie User Stories, Tasks, Zeit- und Prioritätenabschätzung, Iterationen und ein mögliches Project Board exemplarisch dargestellt. Das Beispiel kann als Orientierung genutzt werden, um agile Methoden zu verstehen und in eigenen Projekten umzusetzen.

Keywords: Projektunterricht, Agile Methoden der Softwareentwicklung

1 Einleitung

Eine wesentliche Herausforderung bei der Implementierung innovativer Unterrichtsmethoden stellt das Finden geeigneter Beispiele dar, an denen sowohl die Lehrkraft, als auch die Schülerinnen und Schüler Kernideen und Umsetzungsmöglichkeiten konkret nachvollziehen und erfassen können. Eine solche Innovation stellen im Moment agile Methoden der Softwareentwicklung dar, die Potential besitzen, Probleme zu vermeiden, die Projektunterricht, der herkömmlichen, linearen Prozessmodellen folgt, regelmäßig offenbart. Unter agilen Methoden ist eine Sammlung von Artefakten und Praktiken zu verstehen, die den Projektablauf strukturieren und dabei den Beteiligten konkrete Handlungsoptionen und Kommunikationsanlässe zur Hand geben. Die Praktiken müssen nicht starr verwendet werden, sondern können passend zu dem jeweiligen Projekt neu zusammengestellt werden. Etablierte Frameworks agiler Methoden sind zum Beispiel Scrum, eXtreme Programming oder auch Kanban.

Im Folgenden werden für Unterrichtsprojekte sinnvolle agile Praktiken kurz charakterisiert und am Beispiel des Spiels „Pengu“ exemplarisch umgesetzt. Der Ablauf orientiert sich dabei am agilen Projektmodell für den Informatikunterricht AMoPCE ([RG12], vgl. Abb. 1). Hierbei werden zu Beginn alle bisher bekannten und gewünschten Anforderungen an das System bzw. die Software definiert und in diesem Projekt als sogenannte User Stories festgehalten. Diese als „Geschichten“ beschriebenen Funktionen werden noch weiter hinsichtlich ihrer Umsetzung spezifiziert, Prioritäten und Aufwand abgeschätzt und auf einem Project Board festgehalten. Der weitere Prozess gliedert sich in mehrere Iterationen, wobei jede Iteration ein „Mini-Projekt“ darstellt, in welchem die

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Didaktik der Informatik, Martensstr. 3, 91058 Erlangen
petra.kastl@fau.de, silva.maerz@fau.de, ralf.romeike@fau.de

Phasen des Planens, Entwerfens, Implementierens und Testens jeweils abschließend durchlaufen werden. Innerhalb der Iterationen wird zu Beginn jeder Unterrichtseinheit ein Standup-Meeting durchgeführt, die Projektorganisation wird unterstürzt durch ein Project-Board.

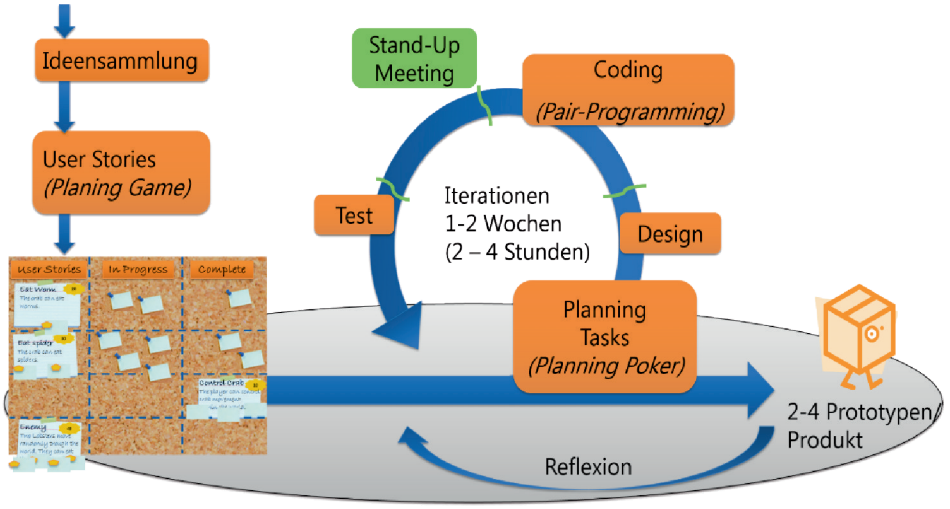

Abb. 1: Agiles Projektmodell für den Informatikunterricht AMoPCE

2 Agile Methoden am Beispiel von „Pengu“

Das Spiel „Pengu“ ist ein Jump ’n’ Run-Spiel, bei welchem es die Aufgabe des Spielers ist, eine Spielfigur auf einer bewegten Wolke über einen Abgrund zu steuern (siehe Abb. 2). Umgesetzt wird dieses Szenario in diesem Beispiel mit Greenfoot. Greenfoot stellt auf der Programmiersprache Java basierende Miniwelten zur Verfügung, welche insbesondere für zweidimensionale Spiele und Simulationen geeignet sind. Greenfoot ermöglicht Programmieranfängern, die objektorientierte Programmierung auf interaktive Weise kennenlernen.

Im Folgenden soll nun die Spielidee konkret mit Hilfe agiler Methoden umgesetzt werden. Hierzu wird das Szenario zuerst in sogenannte User Storys und Tasks aufgeteilt. Anschließend wird das weitere Projektvorgehen beschrieben.

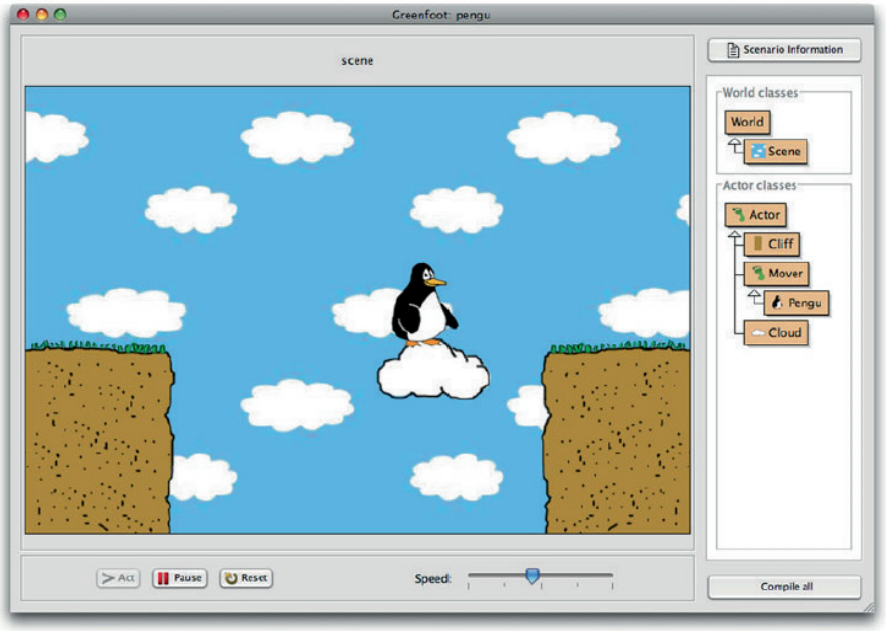

Abb. 2: Spiel „Pengu“ in Greenfoot

2.1 User Stories und Tasks

User Stories beschreiben in kurzer Form Funktionalitäten und Fähigkeiten der zu entwickelnden Software, die dem Nutzer zur Verfügung stehen sollen. Jede User Story beschränkt sich dazu auf eine konkrete Aktivität und wird aus Sicht des Kunden beschrieben. Entsprechend sollten sie so kurz sein, dass sie auf eine Karteikarte passen und auch ohne Programmierkenntnisse verständlich sind. Auf den Karteikarten kann später auch der prognostizierte Aufwand sowie die Priorität vermerkt werden.

Die Erstellung der User Stories erfolgt unter Einbeziehung des Auftraggebers und erfordert domänenspezifisches Wissen aus dem Kontext der zu erstellenden Software. So ist es in unserem Beispiel erforderlich, dass das Team eine Vorstellung vom Spielfluss und Möglichkeiten solcher Jump-and-Run-Spiele besitzt, um die Anforderungen klar aufzustellen und auch eigene Ideen mit einzubringen.

Eine User Story besteht aus einem Titel, einer Beschreibung, einer zeitlichen Abschätzung und einer Priorität. Sie ist nicht länger als drei Sätze, verwendet keine technischen Begriffe und spezifiziert auch keine Werkzeuge. Auch im professionellen Bereich werden für User Stories häufig Karteikarten verwendet, da diese ein passendes Format haben und gut zu handhaben sind.

Alle User Stories zusammen spezifizieren das finale Produkt. Deshalb bilden diese auch die Grundlage für die Präzisierung des Projektziels und die Verständigung mit dem Kunden und der Projektgruppe. Gemeinsam mit dem Kunden werden anschließend die einzelnen Prioritäten festgelegt, aus welchen hervorgeht, welche User Stories zuerst umgesetzt werden sollen und welche ggf. auch nicht umgesetzt werden (vgl. Abschnitt 2.3).

In unserem „Pengu"-Beispiel helfen die User Stories, das umfangreiche Spiel in mehrere Teilbereiche zu gliedern und aufzuteilen. Weiterhin wird durch User Stories leichter deutlich, ob etwas vergessen wurde. Als Grundsatz sollte für jede Rolle (hier z. B. Pinguin bzw. Wolke) und jede einzelne Aufgabe (hier z. B. Pinguin kann sich bewegen) eine eigene User Story geschrieben werden (vgl. [Wo11]).

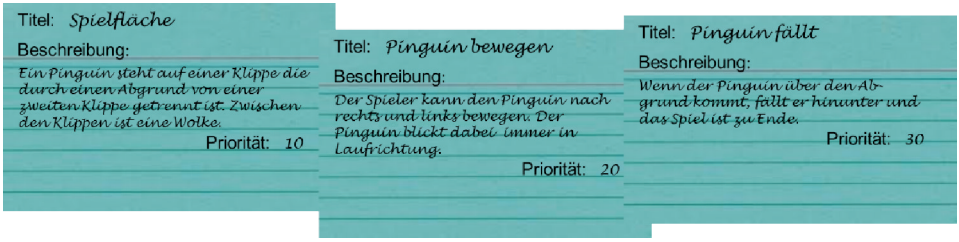

Abb. 3: Die ersten drei User Stories von „Pengu"

Die ersten drei User Stories dienen in unserem Beispiel der Grundfunktionalität des Spiels: „Spielfläche", „Pinguin bewegen" und „Pinguin fällt". Weitere User Stories beinhalten mögliche Ausbaufunktionen. Abb. 3 zeigt die entsprechenden Beschreibungen der User Stories, sowie die Prioritäten. Da alle drei User Stories für das Spiel grundlegend sind, wurden sehr hohe Prioritäten festgelegt (vgl. Abschnitt 2.2), aus denen auch implizierte Abhängigkeiten ersichtlich sind (ehe das Szenario steht, kann keine Aktivität implementiert werden). Nach der Festlegung der User Stories kann mithilfe des Planungsspiels (Planning Poker) der zeitliche Aufwand der Umsetzung der User Stories abgeschätzt werden. Möglichkeiten zur Umsetzung werden im nachfolgenden Abschnitt beschrieben.

Nach der ersten Zeitabschätzung und Priorisierung werden die User Stories der ersten Iteration in Tasks überführt. Ein Task ist eine grobe Beschreibung eines Arbeitspakets, das von einem Entwickler oder Entwicklerpaar (beim Pair Programming) umgesetzt werden kann. Entsprechend kann eine User Story als eine Sammlung von Tasks angesehen werden. Während User Stories die Ziele des „Pengu"-Spiels aus Sicht des Kunden beschreiben, müssen die Schüler nun ihre Perspektive ändern und die Ziele aus Sicht des Entwicklers betrachten. Beim Aufteilen der User Stories zu Tasks sind bereits verschiedene Designentscheidungen zu treffen. Ein prototypisches Beispiel, wie eine umfangreichere User Story in Tasks überführt werden kann, zeigt Abb. 4. Die Überführung der ersten drei User Stories von „Pengu" in Tasks ist in Abb. 5 dargestellt. Hierbei wird deutlich, dass die Tasks bereits konkrete Umsetzungsangaben beinhalten, woran die

Entwicklersicht deutlich wird. Tasks dürfen auch konkrete, technische Begriffe oder kurze Quellcodehinweise enthalten. Notiert werden die Tasks in der Regel auf Klebezetteln (Post-Its). Jeder Task erhält eine eigene zeitliche Abschätzung, die wieder über ein Planungsspiel erarbeitet werden kann. Bei größeren Projekten bietet es sich außerdem an, das Namenskürzel der Schüler zu notieren, die diesen Task bearbeiten. Lehrer und Schüler können hierdurch genau sehen, welcher Schüler welchen Tasks übernommen hat. Weiterhin wird es für die Schüler leichter, am Ende eine Soll-Ist-Gegenüberstellung vorzunehmen.

Abb. 4: Aufteilen von User Stories in Tasks

2.2 Prioritäten- und Zeitabschätzung

Die erste Prioritätenschätzung wird in der Regel gemeinsam mit dem Kunden vorgenommen, wofür sich im Unterricht der Lehrer anbietet. Hierbei ist zu berücksichtigen, was dem Kunden besonders wichtig ist und was gegebenenfalls nur ein „nice to have" wäre. Hier kommt der flexible Charakter des agilen Modells zum Tragen: Entsprechend dem agilen Manifest (vgl. [Be01]) ist es wichtiger, auf Veränderungen reagieren zu können, als nur strikt einem Plan zu folgen. Da – wie jeder Lehrer im Unterricht regelmäßig erlebt – organisatorische, technische oder gesundheitliche Probleme den geplanten Unterrichtsverlauf stören können, wird durch die Festlegung und Umsetzung höher priorisierter Aufgaben ermöglicht, dass selbst bei reduzierter Projektlaufzeit ein lauffähiger und getesteter Prototyp vorhanden ist – und selbst die Prioritäten im Laufe des Projekts noch angepasst werden können. Die Prioritäten werden in Vielfachen von 10 angegeben,

wobei 10 die höchste Priorität hat, bei unserem Spiel ist es die Spielfläche, und nach oben hin die Priorität abnimmt. Verschiedene User Stories können auch die gleiche Priorität besitzen.

Die zeitliche Einschätzung von Tasks wird von den Schülern eigenständig ohne Kunden vorgenommen, es ist aber wichtig, dass alle Teammitglieder in den Schätzprozess mit eingebunden sind. Als Grundlage sollten sinnvolle Zeiteinheiten gewählt werden. Im professionellen Bereich werden Tage als Basis genommen, in Unterrichtsprojekten haben sich Einheiten von 15 Minuten als sinnvoll und gut handhabbar erwiesen. Eine gute Möglichkeit, Vorgaben durch nur wenige erfahrene Schüler zu vermeiden und alle Schüler in die Kommunikationsprozesse einzubinden, stellt das Planungsspiel (Planning Poker, vgl. [PM08]) dar. Hierbei legen sich alle Teammitglieder unter Verwendung eines speziellen Kartenspiels verdeckt fest, welcher Aufwand ihrer Meinung nach für einen bestimmten Task erforderlich ist. Nach Aufdecken der Einschätzungen müssen die extremen Schätzungen begründet werden und es wird versucht, in der Diskussion einen Konsens zu finden. Alternativ wird ein Durchschnittswert der Schätzungen herangezogen. Der Aufwand einer User Story ergibt sich aus den addierten Schätzwerten der einzelnen Tasks. Die Diskussion zur Einschätzung ist gerade für unerfahrene Schüler sehr wichtig, da die Schüler hierbei auch mögliche Strategien austauschen und voneinander lernen können. Gleichzeitig ist dies eine gute Kontrolle, ob alle das gleiche Verständnis von dem haben, was sie mit den Tasks geplant haben. Wenn die Schätzungen auseinander gehen, werden ggf. Unklarheiten in der Diskussion aufgedeckt.

2.3 Iterationen und Prototypen

In Schulsoftwareprojekten stellt sich das Planen und Durchführen langer Projektphasen regelmäßig als Hauptschwierigkeit heraus. Insbesondere ist es schwierig, die Motivation der Schüler aufrecht zu erhalten, wenn über lange Strecken der Erfolg nicht sichtbar wird und das Produkt erst in späten Implementierungsphasen oder zum Ende des Projekts ausprobiert werden kann. Durch die iterative Entwicklung haben die Schüler nun die Möglichkeit, Prototypen des „Pengu“-Spiels in verschiedenen Entwicklungsphasen auszuprobieren und dabei zu überprüfen, ob die Teilziele erreicht wurden.

Im „Pengu“-Beispiel wird die erste Iteration (vgl. Abb. 5) bestimmt durch die Herstellung der Grundfunktionalitäten des Spiels. Der Vorteil der iterativen Vorgehensweise wird unmittelbar deutlich: Bereits früh im Projekt ist das Spiel „spielbar“, der erste Erfolg kann bereits getestet und gegenüber Mitschülern und dem Lehrer demonstriert werden. Sogar das Hinzufügen weiterer Spielideen (Features) ist vor einer Iteration noch möglich, was bei linearen Vorgehensmodellen nahezu ausgeschlossen wäre. Zum Abschluss jeder Iteration ist es möglich, zu reflektieren, wie gut der Prozess bis dahin gelaufen ist und ob sich das Team auf dem richtigen Weg befindet.

In den weiteren Iterationen kommen nun nach und nach, der zuvor bestimmten Priorität entsprechend, weitere Funktionalitäten hinzu, wie sie auf den User Stories festgehalten wurden, z. B. das Bewegen der Wolke, Springen, Überqueren des Abgrunds in Iteration

2, Punktezähler, Sternenhimmel und Sternesammeln in Iteration 3 sowie das Berücksichtigen verschiedener Leben, Gewinnen und Spielende in Iteration 4. Erweiterungen des Spiels, wie bspw. die Eiszapfen, dargestellt in Abb. 4, können auch später noch hinzugefügt werden.

So stellt sich jede Iteration als Mini-Projekt dar, in welchem jede Phase des Softwareentwicklungsprozesses (Anforderungen, Entwurf, Implementieren und Testen) einmal durchlaufen wird und aufgrund der geringeren Komplexität einfacher zu handhaben ist. Hierdurch erhalten die Schüler die Gelegenheit, den gesamten Prozess mehrfach in einem Projekt zu durchlaufen, daran zu lernen, bisherige Erfahrungen zu reflektieren und verschiedene Aufgaben im Team zu übernehmen.

Abb. 5: Tasks für die erste Iteration

2.4 Project Board und Standup-Meetings

Das Project Board ist der zentrale Informations- und Organisationsort des Projekts. Es visualisiert die Ziele und den Status der aktuellen Iteration und unterstützt zielgerichtete Diskussionen anhand der angebrachten User Stories und Tasks. Hier finden sich im Wesentlichen drei Bereiche: Der Vorrat der User Stories mit Tasks, Tasks in Bearbeitung, sowie erledigte Tasks und User Stories. Zu Beginn einer Iteration befinden sich alle Karten auf der linken Seite. Sobald ein Schüler oder Schülerpaar mit einer Aufgabe beginnt, wird ein Post-it mit dem entsprechenden Task aus der Vorratsspalte genommen, mit Namenskürzel versehen und in die Spalte „in Bearbeitung“ gehängt. Sobald der Task bearbeitet wurde, wird er auf „erledigt“ weitergehängt. Ein Beispiel für ein Project Board in der ersten Iteration des „Pengu“-Spiels zeigt Abb. 6.

Am Project Board finden auch die regelmäßigen Standup-Meetings statt, in welchen zu Beginn einer Unterrichtsstunde organisatorische Aspekte der Projektdurchführung be-

sprochen werden. Sie stellen eine geschickte Möglichkeit dar, den Stand des Projekts, das Geleistete der letzten Unterrichtsstunde und Probleme, aber auch die Ziele des Tages zu besprechen. In der Regel sind diese Meetings so kurz, dass es sich gar nicht lohnt, sich erst hinzusetzen. Durch das Stehen bemüht sich auch jeder, sich kurz zu fassen und auf das Wesentliche zu beschränken. Reihum beantwortet hier jeder Schüler die Fragen: Was habe ich seit dem letzten Mal getan? Was werde ich heute in Angriff nehmen? Welche Probleme hatte ich oder sehe ich auf mich zukommen?

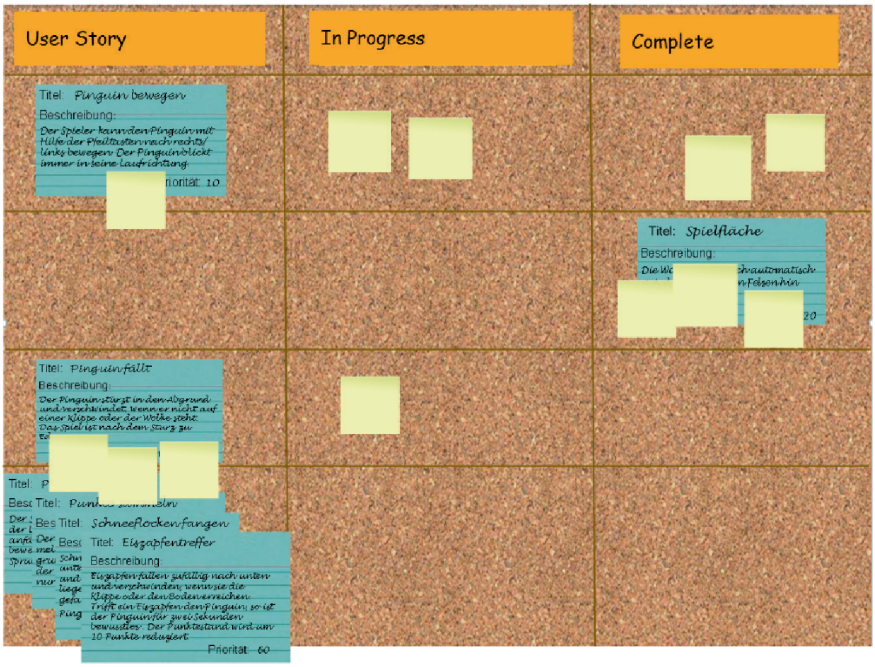

Abb. 6: Das Project-Board zum „Pengu“-Spiel in der ersten Iteration

3 Praxiserfahrungen und Diskussion

Das beschriebene Beispiel wurde erstmals in einem Workshop zu agilen Methoden für den Informatikunterricht von Lehrern ausprobiert und anschließend an verschiedenen Schulen in der Unterrichtspraxis eingesetzt (vgl. verschiedene Beiträge in diesem Band). Dabei hat sich gezeigt, dass das Spiel als Unterrichtsgegenstand sehr motivierend war und sich die Methoden als sehr flexibel erweisen und für Projekte unterschiedlichster Ausprägungen geeignet sind. So können sowohl die Lerngruppen, Vorkenntnisse als auch zeitlichen Gegebenheiten ganz unterschiedlich ausgestaltet werden, weshalb in diesem Beispiel auf die Konkretisierung der Rahmenbedingungen verzichtet wurde.

Schwierigkeiten ergaben sich regelmäßig aus der Notwendigkeit, die agilen Praktiken zuerst selbst erfassen zu müssen. Wie bspw. User Stories formuliert sein sollen und wie man diese in Tasks überführt, stellte sich mangels Erfahrung im ersten Workshop als eine Herausforderung dar. Mit diesem Beispiel ist nun eine Vorlage gegeben, die zur Orientierung für das Erlernen agiler Praktiken und die Durchführung agiler Softwareentwicklungsprojekte im Informatikunterricht dienen kann.

Literaturverzeichnis

[Be01] Beck, K.; Beedle, M.; Bennekum, A.v. et al.: Manifesto for Agile Software Development. http://www.agilealliance.org/the-alliance/, abgerufen am 10.04.2015.

[PM08] Pilone, D.; Miles, R.: Softwareentwicklung von Kopf bis Fuß. Beijing: O'Reilly, 2008.

[RG12] Romeike, R.; Göttel, T.: Agile Projects in High School Computing Education – Emphasizing a Learners ' Perspective. In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE'12): ACM, Hamburg, 2012.

[Wo11] Wolf, H.: Agile Softwareentwicklung (2., aktualisierte und erweiterte Aufl). Heidelberg: Dpunkt.Verlag, 2011.

Anhang: User Stories des „Pengu“-Spiels

Titel: Spielfläche

Beschreibung:

Ein Pinguin steht auf einer Klippe die durch einen Abgrund von einer zweiten Klippe getrennt ist. Zwischen den Klippen ist eine Wolke.

Priorität: 10

Titel: Pinguin fällt

Beschreibung:

Wenn der Pinguin über den Abgrund kommt fällt er hinunter und das Spiel ist zu Ende.

Titel: Pinguin überquert Abgrund

Beschreibung:

Der Pinguin kann auf der Wolke stehen und bewegt sich mit ihr mit. Der Spieler kann ihn auf der Wolke wie auf den Klippen nach rechts und links bewegen.

Priorität: 40

Titel: Punktezähler

Beschreibung:

Das Spiel hat einen Punktezähler, den die Welt verwaltet.

Priorität: 55

Titel: Sternenhimmel

Beschreibung:

Es erscheinen Sterne am Himmel, stehen dort unterschiedlich lang und verschwinden dann wieder.

Priorität: 60

Titel: Pinguin bewegen

Beschreibung:

Der Spieler kann den Pinguin nach rechts und links bewegen. Der Pinguin blickt dabei immer in Laufrichtung.

Priorität: 20

Titel: Wolkenbewegung

Beschreibung:

Die Wolke bewegt sich automatisch zwischen den Klippen hin und her.

Priorität: 30

Titel: Pinguin springt

Beschreibung:

Der Spieler kann den Pinguin springen lassen. Wenn der Pinguin fällt reagiert er nicht. Während des Sprungs behält der Pinguin eine anfängliche horizontale Bewegung und seine Blickrichtung bei.

Priorität: 50

Titel: Pengu gewinnt

Beschreibung:

Auf der rechten Klippe steht eine Tür. Solange der Spieler mehr als 15 Punkte hat ist sie offen, sonst ist sie geschlossen. Erreicht Pengu die Tür, wenn sie offen ist ertönt eine Fanfare und der Spieler hat gewonnen.

Ganzjähriger Projektunterricht mit agilem Framework

Ulrich Kiesmüller[1] , Petra Kastl[2] und Ralf Romeike[3]

Abstract: In diesem Beitrag werden zwei jeweils achtmonatige Unterrichtsprojekte zweier 10. Klassen eines bayerischen Gymnasiums vorgestellt. Über die gesamte Zeit entwickelten Gruppen von je fünf bis neun Schülerinnen und Schülern mit der Programmiersprache Java ihr eigenes Softwareprojekt und erarbeiteten sich dabei informatische Konzepte der objektorientierten Programmierung und Modellierung. Zur Unterstützung wurden geeignete agile Praktiken ausgewählt und jeweils zeitverzögert durch weitere ergänzt. Die vorgenommene Anpassung des agilen Modells an den Kontext, die praktische Umsetzung und Beobachtungen werden im vorliegenden Beitrag beschrieben. Sie werden kontrastiert zu den Erfahrungen aus den Vorjahren, in denen nach dem Wasserfallmodell vorgegangen wurde. Abschließend werden wesentliche Erkenntnisse und Erfahrungen, die in die Weiterentwicklung des agilen Modells fließen, zusammengestellt.

Keywords: Einsatz agiler Methoden der Softwareentwicklung im Informatikunterricht, Projektunterricht

1 Objektorientierung und Softwareentwicklung im Unterricht

Grundlagen der objektorientierten Modellierung und Programmierung sind für die naturwissenschaftlich-technologische Ausbildungsrichtung an bayerischen Gymnasien im Lehrplan der 10. Jahrgangsstufe [IS03] verankert. Als Abschluss ist dort ein kleines Softwareprojekt vorgesehen, um den Lernenden zu vermitteln, dass man umfangreiche Aufgaben nur mit sorgfältig geplanter Teamarbeit, strukturiertem Vorgehen und basierend auf fachlichem Wissen lösen kann. Hierbei geben der bayerische Lehrplan und die gängigen Schulbücher dem Wasserfallmodell den Vorzug. Über diese vom Lehrplan geforderten Ziele hinaus sind mir als Lehrkraft auch die Berücksichtigung von Aspekten wie selbstständiges Arbeiten, Kreativität und kritische Reflexion [HNR07] wichtig. Außerdem soll das Projekt Sozialkompetenzen wie Kommunikation und die Fähigkeit zur Entscheidungsfindung in einer Projektgruppe sowie einen konstruktiven Umgang mit Konflikten [Gu08] fördern. Überdies möchte ich den Lernenden mit der Projektarbeit ein modernes, zeitgemäßes Bild vom Beruf eines Informatikers vermitteln [GR13].

In der praktischen Umsetzung erlebte ich wiederholt, dass Schülerinnen und Schüler am Ende der 10. Jahrgangsstufe in Projekten zwar sehr motiviert sind, aber in den zehn Unterrichtsstunden, die der Lehrplan vorsieht, keine brauchbaren oder gar spannenden Produkte gemeinsam planen, entwickeln, vorstellen und reflektieren können. Dafür ist die

[1] Simon-Marius-Gymnasium Gunzenhausen, Simon-Marius-Straße 3, 91710 Gunzenhausen, kiesmueller@simon-marius-gymnasium.de
[2] Friedrich-Alexander-Universität Erlangen-Nürnberg, Didaktik der Informatik, petra.kastl@fau.de
[3] Friedrich-Alexander-Universität Erlangen-Nürnberg, Didaktik der Informatik, ralf.romeike@fau.de

Zeit zu knapp bemessen. Deshalb habe ich in den letzten Jahren die Projektphase erheblich ausgedehnt und weitere Lehrplaninhalte integriert. Für den dazu notwendigen fachlichen Input wurde die Projektarbeit der Schülerinnen und Schüler regelmäßig meist zu Beginn jeder Doppelstunde unterbrochen. Problematisch bei diesem Vorgehen war, dass die Lernenden zu Beginn des Projekts weder über die fachlichen, planerischen und sozialen Fähigkeiten und Fertigkeiten verfügten, die ein lineares Prozessmodell wie das Wasserfallmodell voraussetzt, noch die Schritte einer strukturierten Vorgehensweise in der Softwareentwicklung kannten. Deshalb waren die Jugendlichen immer stark auf Unterstützung angewiesen. Für die Lehrkraft bedeutete es einen enormen Betreuungsaufwand, wobei die Unterstützung noch anspruchsvoller wurde, wenn jede Projektgruppe ihr eigenes Thema wählen konnten. Entsprechend langsam war der beobachtbare Projektfortschritt. Andererseits erhöhte ein eigenes Thema die Motivation und das Durchhaltevermögen der Lernenden insbesondere in der langen Planungsphase. Und so waren die Rückmeldungen der Lernenden am Ende des Schuljahres zwar großteils positiv und sie waren stolz auf ihr Produkt. Das lange Warten auf Unterstützung wurde aber regelmäßig von den meisten Projektbeteiligten bemängelt.

Im Einsatz agiler Methoden sah ich die Möglichkeit, es den Lernenden durch iteratives Vorgehen zu ermöglichen, mit vorhandenen fachlichen, sozialen sowie organisatorischen Fähigkeiten und Fertigkeiten, unterstützt durch ausgewählte agile Praktiken und Artefakte, selbstorganisiert loszulegen und ihre Kompetenzen in jeder Iteration auszubauen. Die Rolle der Lehrkraft wandelte sich hierbei vom „Fragenbeantworter“ und „Fehlersucher“ hin zu 50% Coach und 50% Beobachter (siehe Kapitel 5). Sehr gut konnte ich den fachlichen Lernfortschritt und die positive Entwicklung sozialer und organisatorischer Fähigkeiten bei den Schülerinnen und Schülern beobachten. Von zu langen Phasen des Wartens auf Unterstützung war nicht mehr die Rede. Obwohl ich kaum noch helfen musste, gaben die Lernenden in der Rückmeldung an, dass sie sich gut betreut fühlten.

2 Rahmenbedingungen

In diesem Artikel werden Erfahrungen aus zwei 10. Klassen eines naturwissenschaftlich-technologischen Gymnasiums (NTG) in Bayern vorgestellt. Die Klassen hatten beide ca. 25 Schülerinnen und Schüler und wurden jeweils in Doppelstunden unterrichtet, die im selben Computerraum mit 20 Einzelrechnern stattfanden. Die Größe des Raumes bot jeder Gruppe Platz für ihr Project Board sowie die davor stattfindenden Diskussionen und Planungen, ohne dass sich die Schülergruppen dabei gegenseitig störten. Die Vorkenntnisse im Bereich der Objektorientierung aus der 6. Jahrgangsstufe und und der Algorithmik aus der 7. Jahrgangsstufe waren bei den meisten Schülerinnen und Schülern gering. Allerdings verfügten beide Klassen über einige Projekterfahrung, weil sie in den vorangegangenen Jahren mehrere Projekte erfolgreich durchgeführt hatten. In der 10. Jahrgangsstufe gilt es wie oben angeführt die Grundlagen der objektorientierten Modellierung und Programmierung unter Verwendung von Java zu vermitteln. Als Entwicklungsumgebung wurde dabei BlueJ [KB09] verwendet. Als weitere Hilfsmittel standen den Lernenden das Buch *Java ist*

auch eine Insel [Ul14] zur Verfügung sowie die Java-Klasse ZEICHENFENSTER[4], die es erlaubt, einfache geometrische Objekte in einigen wenigen Farben graphisch darzustellen.

3 Einsatz und Anpassung eines agilen Frameworks

3.1 Agile Praktiken in der Vorbereitung des Projekts

Vor dem Projektstart werden die grundlegenden Voraussetzungen bezüglich der objektorientierten Modellierung und Programmierung sowie des Einsatzes geeigneter Werkzeuge vermittelt. Parallel dazu werden erste Elemente des agilen Modells für Projekte im Informatikunterricht (AMoPCE) [RG12] angepasst und in den Unterrichtsverlauf integriert. In diesem Kapitel beschreibe ich das Vorgehen und dahinter stehende Intentionen exemplarisch. Nach einem Theorie-Input zur Erstellung von Klassendiagrammen erhalten die Schülerinnen und Schüler eine Aufgabe, die sie in Kleingruppen kooperativ planen und modellieren. Da sie später im Projekt mit der Klasse ZEICHENFENSTER arbeiten werden, bieten sich Themen wie das Erstellen eines Szenenhintergrunds oder von Figuren für ein Spiel an, die aus einfachen geometrischen Objekten zusammengesetzt sind. Die Planung erfolgt in einem **Stand-Up-Meeting** vor dem **Project Board** der Gruppe, das hier nur eine freie Planungsfläche ist. Die Lernenden üben dabei im Stand-Up-Meeting Diskussionen effektiv und konzentriert zu führen, Absprachen zu treffen, Probleme zu identifizieren und sich auf einen gemeinsamen Plan zu einigen. Ihr Project Board ist für sie von Beginn an ein zentraler Arbeits- und Planungsbereich. In der Rolle eines geeignet angelegten Kunden, der **Kundengespräche** mit den einzelnen Gruppen führt, integrierte ich wesentliche Aspekte einer Anforderungsermittlung in die Planung. Aufgabe der Teams ist es, durch gezielte Fragen Kundenwünsche zu präzisieren, dem Kunden ihren geplanten Entwurf zu „verkaufen“ und die Ergebnisse in der Modellierung umzusetzen. Bei der anschließenden arbeitsteilig durchgeführten Implementierung wird in das Werkzeug BlueJ eingeführt, die Klasse ZEICHENFENSTER vorgestellt und zum ersten Mal Quelltext zusammengeführt. Im weiteren Verlauf werden Grundlagen der objektorientierten Programmierung und der algorithmischen Grundstrukturen in mehreren Kleinstprojekten vermittelt. Die Arbeitsaufträge werden in Form von **User Stories** (also Funktionalitäten aus Kundensicht) gestellt. Zu Beginn sind diese sehr präzise formuliert und durch Teilaufträge in Form von **Tasks** (also zu erledigende Aufgaben aus Entwicklersicht) ergänzt. Gegen Ende werden User Stories auch gezielt offener formuliert („Ich möchte, dass sich ein Ball quer über den Bildschirm bewegt.“, „Ich möchte, dass sich ein Kreis in einem Kreisring auf dem Bildschirm bewegt.“), um Spielraum für eigene Interpretationen zu schaffen und um die Lernenden die Tasks selbständig identifizieren und formulieren zu lassen. So führte ich sie schrittweise an den Perspektivwechsel von Kunden- zu Entwicklersicht heran. Die Bearbeitung der Aufträge erfolgte großteils in Gruppen und die Planung fand weiterhin in Stand-Up-Meetings vor dem Project Board statt. Für die Implementierung bilden die Schülerinnen und Schüler Paare und nehmen abwechselnd die Rolle des *Drivers*[5] und des *Navigators*[6]

[4] erstellt von M. Kölling, B. Quig und Ch. Heidrich

[5] bedient die Tastatur, schreibt den Code und denkt dabei laut

[6] behält das große Ganze im Auge, schlägt Alternativen vor und spricht Fehler an

ein. Aus didaktischer Sicht ist dieses Pair-Programming für mich interessant, weil Lernende dadurch passiv voneinander lernen und üben, ihre Codiertätigkeiten in Worte zu fassen und verschiedene Codierstile zu diskutieren. Themen und Aufgabenstellungen der Kleinstprojekte wurden so gewählt, dass User Stories, Tasks und erstellter Code für die Großprojekte angepasst und dort integriert werden konnten. Die in dieser Phase gebildeten Gruppen hatten stets wechselnde Zusammensetzungen, damit sich keine eingespielten Rollenverteilungen herausbildeten und die Lernenden unterschiedliche Herausforderungen kooperativen Arbeitens zu bewältigen lernten. Die Schülerinnen und Schüler konnten Kundengesprächstermine mit mir vereinbaren, wenn sie Unterstützung benötigten.

3.2 Das Projekt

Um den Lernenden spielerisch eine Idee von agiler Projektarbeit zu vermitteln und ihnen Mut zu machen, wird vor dem Einstieg ins Projekt ein „Warm-Up-Spiel“[7] durchgeführt. Hierbei erfahren die Schülerinnen und Schüler die Bedeutung von gemeinsamen Absprachen, Vorteile kurzer, iterativer Entwicklungsphasen (flexible Reaktion auf Änderungen, stete Verbesserung der eigenen Performanz, Sinn einer Reflexion) und die motivierende Wirkung von Zielsetzungen, die in kurzer Zeit erreichbar sind. Stimmen aus dem Kreis der Lernenden nach der Durchführung waren unter anderem: „Noch nie habe ich mich für etwas so reingehängt.“ „Hätte nicht gedacht, dass wir uns so steigern können.“ „Schade, dass wir nicht noch weiter gemacht haben, da wäre noch mehr drin gewesen.“ Für die sich bis zum Schuljahresende erstreckende Projektarbeit dürfen sich die Lernenden eigene Themen wählen. Die Gruppen bilden sich dann entsprechend der fachlichen Interessen.

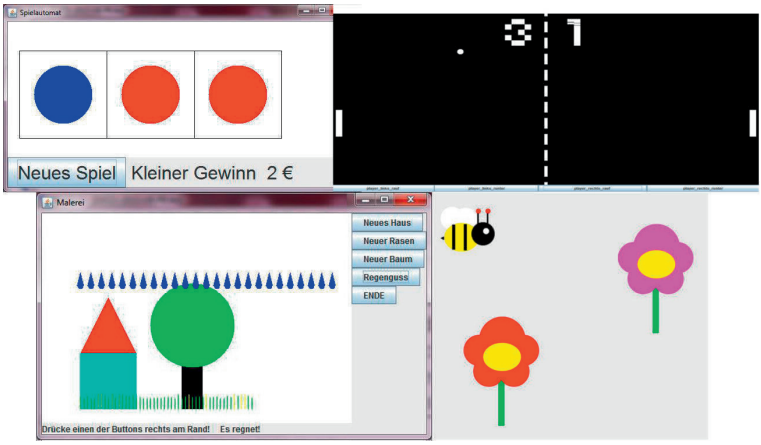

Abb. 1: Anreize zur Themenwahl: Spielautomat, Arcade-Game, bewegte Grafiken

Durch Präsentation einiger, in der zur Verfügung stehenden Zeit umsetzbarer, Beispiele (vgl. Abbildung 1) wird ihnen eine Entscheidungshilfe geboten. Nach der folgenden geheimen Abstimmung der Lernenden bezüglich eines Wunschthemengebiets ließen sich die

[7] http://borisgloger.com/scrum/materialien/tools/

Klassen jeweils problemlos in drei etwa gleich große Interessensgruppen teilen. Die themengebundene Zusammensetzung der Projektgruppen führt dazu, dass in vielen Fällen nicht nur „eingespielte Teams“ zusammenarbeiten und sich die Schülerinnen und Schüler mit für sie „neuen“ Gesprächs- und Teampartnern zurechtfinden müssen.
Die Teams treffen sich zu einer ersten Besprechung in einem Stand-Up-Meeting an ihrem Project Board, das ihnen bis Schuljahresende exklusiv zur Verfügung steht. Die Lehrkraft stößt zu jedem Team, verfolgt die Gespräche einige Zeit, schlüpft dann in die Rolle des Kunden, um die Wünsche zu kanalisieren und verdeutlicht als Berater erste erreichbare Etappenziele. Dieses Vorgehen wiederholt sich im weiteren Projektverlauf, wobei die Kontrolle über das Projekt sobald wie möglich vollständig der Gruppe überlassen wird. Anschließend greift die Lehrkraft nur noch dann ein, wenn die Schülerinnen und Schüler sich deutlich zu hohe Anforderungen stellen oder völlig falsche Wege bei der Implementierung beschreiten wollen. Die Produktfunktionalitäten, die Ziel der jeweiligen Etappe sind, werden als User Stories formuliert und in Reihenfolge ihrer Priorität mit den Namen der Bearbeitenden versehen und an das Project Board (siehe Abbildung 2) geheftet.

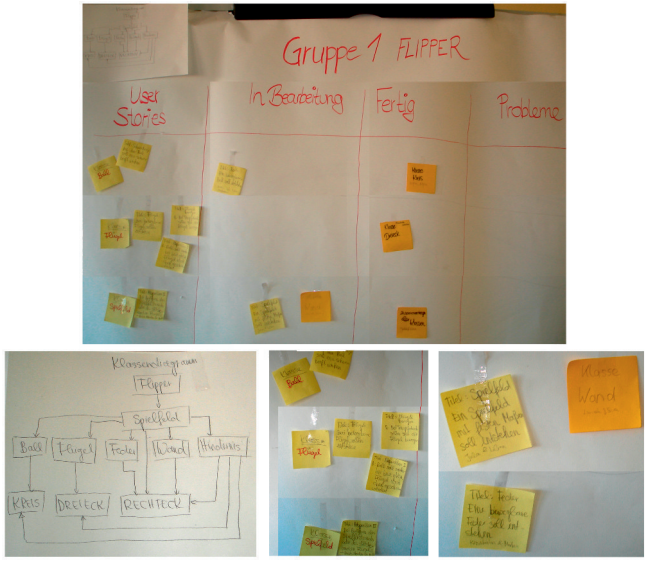

Abb. 2: Project Board „Flipper“ – Klassendiagramm – User Stories/Tasks geplant/in Bearbeitung

Dabei ergibt sich die Priorität entweder logisch aus den Funktionalitäten oder die Teammitglieder einigen sich darüber, was ihnen wichtiger und was weniger wichtig ist. Das Project Board ist ab jetzt ein Board im engeren Sinn der Softwareentwicklung, mit drei Spalten, für User Stories und dazu geplanten Tasks links, für Tasks in Bearbeitung in der Mitte und für fertige Tasks und abgeschlossene User Stories rechts. Sie bieten allen Beteiligten eine ständig aktuelle Übersicht über den Stand des Projektes. Dies gilt sowohl für die Lernenden, die sich bei auftretenden Fragen bezüglich des Verlaufs oder nächster Schritte gerne auch spontan am Board informieren, als auch für die Lehrkraft, die jederzeit einen Überblick hat, wo sich die einzelnen Gruppen aktuell im Projekt befinden und sogar, was die einzelnen Programming-Pairs gerade implementieren. Die Teams halten diese

Informationen immer aktuell. In der rechten Spalte unserer Project Boards gab es eine separate Zelle für **Probleme**. Hier heften die Schülerinnen und Schüler Zettel mit offenen Fragen und Problemen an, die sie nach der Iteration selbstständig oder mit dem Lehrer zusammen klären möchten. Bis zum Ende der Etappe ist der Ablauf nun so, dass die Teams in einem Stand-Up-Meeting in etwa die nächsten 20 Minuten planen. Die Teammitglieder wählen dazu eine oder mehrere User Stories unter Beachtung der Priorität aus, mit deren Umsetzung das Team ca. 20 Minuten beschäftigt ist und formulieren dazu Tasks. Pro Doppelstunde finden so in der Regel zwei Stand-Up-Meetings statt, auf jeden Fall aber eines zu Stundenbeginn. Die geplanten Tasks werden arbeitsteilig und im Pair-Programming implementiert. Das Project Board halten die Schülerinnen und Schüler dabei aktuell. Einen Task, den ein Pair in Bearbeitung nimmt, versieht es vor dem Umhängen mit Namen, damit die Gruppenmitglieder sich bei Fragen gezielt an das Pair wenden können. Auch die Lehrkraft kann, nach Studieren der in Bearbeitung befindlichen Tasks, die entsprechenden Schülerinnen und Schüler direkt ansprechen und sich den Arbeitsstand zeigen lassen. Treten während der arbeitsteiligen Phase dringend durch das ganze Team zu klärende Fragen oder Probleme auf, rufen die Schülerinnen und Schüler ein spontanes Stand-Up-Meeting aus. Gegen Ende der Doppelstunde wird der Code eines Teams zusammengeführt, getestet und falls nötig korrigiert. Ziel ist es, nach jeder Doppelstunde eine getestete und lauffähige Programmversion zu haben. Die nächste Doppelstunde beginnt dann wieder mit einem Stand-Up-Meeting, in dem die Teams erledigte Aufgaben der letzten Stunde rekapitulieren, eventuelle Probleme ansprechen und die nächste Arbeitsphase planen.
In diesen Ablauf wurde der Lerninhalt der 10. Jahrgangsstufe integriert. In den bisherigen Durchläufen zeigte sich, dass die einzelnen Teams trotz unterschiedlicher Themen und individuellem Arbeitstempo oft nahezu zeitgleich an bestimmte Problemstellungen gelangen, z. B. „wir bräuchten etwas, um viele gleichartige Dinge auf einmal anzusprechen“. In diesen Fällen bietet sich eine zentrale Theorie-Input-Phase durch die Lehrkraft an – im genannten Fall zum Thema „(eindimensionale) Felder“. Kommt eine Gruppe deutlich früher zu einer Problemstellung, wird ihr individuell weiter geholfen, um möglichst raschen Projektfortschritt zu gewährleisten. Themen, auf welche die Lernenden nicht von selbst stoßen, können in Kundengesprächen gezielt angeschnitten werden. Beispielsweise wird ein Kundenwunsch nach einer GUI, also einfachen Buttons und Labels mit Begeisterung aufgenommen und mit Eifer angegangen. Hierzu vermittelt die Lehrkraft die Theorie und stellt auch entsprechende Codefragmente zur Verfügung.

4 Beobachtungen und Erfahrungen

Wie in der Einleitung erwähnt, veränderte sich meine Rolle hin zum Coach und Beobachter, da die Schülerinnen und Schüler von Anfang an ihre Projektarbeit selbstständig organisierten. Zu Beginn war dies nur mit Abstrichen „zielorientiert und gründlich“, aber sie behalfen sich, wenn nötig, mit selbst einberufenen spontanen Stand-Up-Meetings und lernten rasch, worauf es bei einer Planung ankommt. Ich griff in der Anfangsphase häufiger als Kunde und/oder Berater unterstützend ein. Später konnte ich als Beobachter durch die starke Betonung der interaktiven Elemente bei den agilen Methoden beispielsweise auch soziale Kompetenzen und deren Entwicklung sehen. Defizite im Bereich der Kommunikationsfähigkeit einzelner Lernender hätte ich früher mit großer Wahrscheinlichkeit nicht

bemerkt, da ich zwar ständig bei einzelnen Teams war, jedoch jeweils nur für kurze Zeit und nur um Fragen zu beantworten. Jetzt konnte ich solche Defizite bemerken und die Entwicklung der Lernenden verfolgen. Dadurch wird die Bewertung und Einschätzung der Einzelleistungen auf einer wesentlich solideren Basis möglich und eine Rückmeldung an die Lernenden über die rein fachlichen Kriterien hinaus möglich.

Einige Tätigkeiten und Prinzipien der Softwareentwicklung, die sonst nur mit Mühe vermittelt werden können, werden von den Schülerinnen und Schülern bei der agilen, iterativen Vorgehensweise als sinnvoll und hilfreich erkannt – denn sie bedürfen keiner zusätzlichen Motivation durch die Lehrkraft. Klassendiagramme beispielsweise erstellen die Teams relativ früh freiwillig und erweitern sie, weil sie ihnen bei der Planung späterer Iterationen, bei der Implementierung sowie beim Testen helfen. Wie wichtig exakte Absprachen im Schnittstellenbereich wie z. B. Namenskonventionen für *Getter* und *Setter* sind, erkennen sie durch die arbeitsteilige Vorgehensweise im Team rasch. Sonst treten beim Testen nach der Zusammenführung Fehler auf oder es muss nachgefragt werden. „Wie heißt dein Getter“ ist zwar schnell durch den Raum gerufen, aber effektiver und ungestörter arbeiten lässt es sich mit verbindlichen Absprachen. Mit zunehmender Programmkomplexität müssen die Lernenden immer häufiger „fremden“ Code lesen und verstehen, um ihn erweitern zu können. Je besser der Code kommentiert ist, umso leichter und schneller geht das. Auch Codefragmente, die Schülerinnen und Schüler sich selbst z. B. mit Hilfe des Openbooks [Ul14] hart erarbeitet haben und die sie gerne voller Stolz weiter geben, führen bei guter Kommentierung zu weniger Nachfragen. Tests am Ende jeder Doppelstunde werden gerne durchgeführt, weil man sehen will, was die anderen Teammitglieder implementiert haben und weil man wieder ein fehlerfreies, lauffähiges Produkt haben möchte. Andere agile Praktiken werden angenommen, aber bis zu ihrer endgültigen Beherrschung benötigen die Lernenden einige Zeit. Die Erstellung kompakter User Stories, die nur aus wenigen einzelnen Tasks bestehen, sowie die Formulierung konkreter, innerhalb einer Arbeitsphase zu bewältigender Tasks, gestaltet sich für die Schülerinnen und Schüler anfangs extrem schwierig. Allerdings helfen theoretische Erläuterungen und Praxisbeispiele hier wenig; erfahrungsgemäßwird diese Technik am Besten durch das Sammeln eigener Erfahrung erlernt: Als zeitversetzt nach einigen Wochen das Planning Poker als Möglichkeit zur Aufwandsabschätzung vorgestellt wurde, hatten die Lernenden bereits ein gutes Gespür für die passende Größe von User Stories und Tasks entwickelt. Die Technik der Aufwandsabschätzung hatte für sie keinen Mehrwert und wurde deshalb verworfen. Ähnliches zeigt sich bezüglich des Planens einer Arbeitsphase. Unabhängig davon, dass mit 20 Minuten ein für Neulinge im Bereich des Programmierens überschaubares Zeitfenster gewählt wurde, gelingt vielen Schülerinnen und Schülern zunächst keine saubere Planung. Stattdessen berufen sie bis zu fünf weitere Stand-Up-Meetings ein, um noch nicht bedachte Probleme zu klären. Nach ein, zwei Doppelstunden jedoch gelingt es den Lernenden problemloser die Arbeitsphase sauber zu planen. Wenn einzelne Teammitglieder vor anderen ihre Aufgaben umgesetzt haben, es aber keine weiteren in der verbleibenden Zeit umsetzbaren Tasks gibt, verbessern die Betroffenen z. B. die Inline-Dokumentation oder testen einzelne Klassen oder das gesamte Projekt.

Etwas schwierig ist es, einen regelmäßigen Wechsel der Rollen beim Pair-Programming zu erreichen. In der Regel übernimmt die besser programmierende Person die Rolle des *Dri-*

vers und die schwächere die Rolle des *Navigators*, obwohl die Lernenden den Sinn eines Rollenwechsels einsehen. Um den Wechsel zu motivieren, wurde von mir nach etwa sechs Wochen der **Truck Factor** eingeführt und ausgerufen. In der Softwaretechnik wird vom Truck Factor gesprochen, wenn es darum geht, dass jeder Mitarbeitende bei spontanem Ausfall eines Teammitglieds vollständig über dessen Aufgaben und Arbeitsstand informiert ist und somit jederzeit die Bearbeitung übernehmen kann. Auch bei schulischen Projekten ist dieser Aspekt nicht von der Hand zu weisen. Immer wieder erkranken Lernende und ihre jeweiligen Partner können dann nicht weiterarbeiten, was zu Verzögerungen in der Projektentwicklung führt. Um die Anforderung zu prüfen, ob sich wirklich jeweils beide Partner eines Programmierteams auf dem gleichen Informationsstand befinden, kann man als Lehrkraft zu einem nicht genauer angekündigten Zeitpunkt den Truck Factor „ausrufen“. Dann wird das angesprochene Paar an zwei verschiedene Rechner gesetzt, die jeweiligen Dateien werden kopiert und beide Partner müssen getrennt voneinander weiterarbeiten. Die Lehrkraft kann sich hierbei von jedem einzelnen erläutern lassen, was seine aktuell zu bewältigenden Aufgaben sind. Anfangs standen die Schülerinnen und Schüler dieser Unterrichtsmethode skeptisch gegenüber, entwickelten aber sehr schnell den Ehrgeiz zu zeigen, dass auch nach Trennung eines Paars beide Beteiligten ohne Nachfragen weiterarbeiten können.

5 Kritischer Vergleich mit klassischen Methoden

Abschließend werden nun die Erfahrungen und Beobachtungen der drei Durchgänge mit projektbasiertem Unterricht kontrastiert mit denjenigen der Vorjahre, in denen ohne Großprojekt arbeitsgleich oder mit vorgegebenem bzw. individuellem Thema, aber nur in Zweiergruppen gearbeitet wurde.

5.1 Sicht der Lehrkraft

Arbeitsgleiches Vorgehen besitzt für die Lehrkraft den Vorteil langfristig möglicher Vorplanung. Der ständig gleiche Unterrichtsverlauf kann aber zur Abstumpfung hinsichtlich der Bedürfnisse und Probleme der Lernenden führen. Außerdem können permanente Wiederholungen demotivierende Effekte hervorrufen. Egal ob diese Methode mit Großgruppen oder mit Programming Pairs durchgeführt wird, ist der Betreuungsaufwand für die Lehrkraft nicht allzu hoch – zumindest ist er kalkulierbar. Individuelle Projekte in Zweiergruppen hingegen fordern die Lehrkraft sowohl in sportlicher als auch informatischer Sicht. Es ist nicht voraussehbar, welches Team zu welchem Zeitpunkt welche Problemstelle erreichen wird. Die Lehrkraft betreibt oft „Turnschuhdidaktik“, indem sie rastlos von einer Gruppe zur nächsten eilt und sich in kürzester Zeit in den jeweiligen Programmcode einlesen und Problemlösungen mit den Lernenden erarbeiten muss. Zusätzlich muss sie noch dafür sorgen, dass alle Theorieinhalte vermittelt werden, was oft dazu führt, dass die Schülerinnen und Schüler „auf Vorrat“ lernen müssen, da sie zum Zeitpunkt der Besprechung gewisser Inhalte diese in keiner Weise in ihrem Projekt benötigen und einsetzen können. Mit Hilfe agiler Methoden lassen sich diese Probleme vermeiden. Die individuelle Projektgestaltung führt zu einer interessanten und abwechslungsreichen, aber auf

Grund der geringen Themenzahl nicht ausufernden Arbeit für die Lehrkraft. Rasch sichtbare Ergebnisse und Anerkennung der Leistungen über die Gruppe hinaus stärken das Selbstbewusstsein der Lernenden und ihr Vertrauen in die eigenen Fähigkeiten. Sie entwickeln den Ehrgeiz, selbstständig Lösungen für ihre Aufgaben zu finden und zeigen dabei ein wesentlich höheres Durchhaltevermögen und deutlich mehr Leistungsbereitschaft als ohne agile Methoden. Auch der Betreuungsaufwand hält sich in Grenzen und ist durch die Verwendung der Project Boards gut planbar. Mit den selbst gewählten Projektthemen erreichen die Lernenden sehr schnell komplexe Fragestellungen wie zum Beispiel: „Wie kann ich eigene Farben definieren?“. Erstens ist es für die Lehrkraft zeitlich nicht möglich, all diese Problemstellungen gleich nach deren Auftreten für die Teams zu lösen. Zweitens wird auch bei Softwareentwicklungs-Großprojekten nicht jede Codezeile direkt erstellt, sondern es werden vielmehr bereits bekannte Teillösungen angepasst und integriert. Diese Technik, mit der die Lernenden bereits im Zusammenhang mit der Java-Klasse ZEICHENFENSTER bei der Erstellung kurzer Programme in Kontakt kamen, wird nun weiter vertieft. Je nach den technischen Gegebenheiten erhalten die Lernenden die Erlaubnis die Online-Version des Openbooks *Java ist auch eine Insel* von Christian Ullenboom [Ull14] zu verwenden oder es wird ihnen die entsprechende Offline-Version zur Verfügung gestellt. Sie dürfen das Buch nicht nur als Informationsquelle verwenden, sondern sich auch bei den Code-Beispielen bedienen, die sie für ihre eigenen Projekte entsprechend anpassen müssen. Die Theorieeinheiten sind zwar hinsichtlich ihres Zeitpunktes nicht langfristig zu terminieren, lassen sich aber gut über das Schuljahr verteilen, so dass auch kleine schriftliche Leistungserhebungen durchgeführt werden können. Die Beurteilung der individuellen Leistungen gestaltet sich, unterstützt durch die Eintragungen an den Project Boards und den Beobachtungen bei Kundengesprächen oder Stand-Up-Meetings, einfacher als bei den zu Beginn des Abschnitts beschriebenen Vorgehensweisen.

5.2 Sicht der Schülerinnen und Schüler

Bei der Projektarbeit in Zweiergruppen bei „klassischem Vorgehen“ kritisieren die Lernenden am meisten die langen Wartezeiten auf Betreuung durch die Lehrkraft. Sie empfinden einerseits die individuell wähl- und gestaltbaren Projektideen als sehr motivierend und insgesamt positiv, sind aber andererseits in Bezug auf den aus oben genannten Gründen sehr langsamen Projektfortschritt unzufrieden. Insbesondere ist es für die Schülerinnen und Schüler frustrierend, dass sie häufig nicht nur wochenlang kaum einen Fortschritt bei ihrer Software sehen können, sondern es bis zum Schuljahresende nicht schaffen, ein funktionsfähiges Produkt erstellt zu haben. Außerdem entstand bei einigen Lernenden der Eindruck, dass ihre individuellen Leistungsbewertung im Verhältnis zu denen ihrer Teampartner nicht gerechtfertigt sei. Auch aus Sicht der Lernenden führt der Einsatz der agilen Methoden dazu, die oben genannten negativen Aspekte zu vermeiden. Sie fühlen sich rundum betreut, sehen selbst an den nach jeder Iteration lauffähigen Programmversionen einen permanenten Projektfortschritt und empfinden die Bewertung ihrer individuellen Leistung durchweg als nachvollziehbar und korrekt. Außerdem meldeten viele Lernende zurück, dass sie durch diese Methode auch „etwas fürs Leben“ gelernt haben.

6 Fazit und Ausblick

Der Einsatz eines agilen Frameworks bei der Gestaltung des Informatikunterrichts mittels eines ganzjährigen Projekts stellt sich aus Sicht aller Beteiligten als positiv dar. Sowohl die Lehrkraft als auch die Lernenden arbeiteten hochmotiviert, die Schülerinnen und Schüler hatten viel Spaß– sicher mitbegründet durch häufige Erfolgserlebnisse – und erlernten vieles, was in den letzten Jahren unerreichbar schien oder gar nicht in Erwägung gezogen wurde. So erschlossen sich fast alle Beteiligten, wie man selbst Farben mit Hilfe der Hexadezimalcodierung definieren kann, einige lernten, wie man mehrere geometrische Grundbausteine zu grafischen Gesamtobjekten kombinieren kann, die dann zum Beispiel mit Farbverläufen gefüllt werden können. Es entstanden Stadtgrafikprojekte, bei denen sich die Sonne auf einer bogenförmigen Bahn über den Himmel bewegt und nachts dann die Lichter in den Fenstern der Häuser angehen. Die Spielautomatenprogramme wurden soweit ausgebaut, dass wirklich ein Hebel wie bei den „One-Arm-Bandits“ betätigt werden musste, um die Walzen zu starten, die während ihres Laufs wechselnde Symbole (verschiedene Früchte und nicht lediglich Kreise) zeigten. Bei Gewinn fielen dann auch Münzen aus dem Geldauswurfschacht. Für eine authentische Darstellung der Arcade-Games wurde für die Punkteanzeige eine 4x8-Punkt-Matrix eingesetzt.

Die Gestaltung des agilen Frameworks wird in weiteren Durchgängen weiter ausgebaut. Die Hilfsprogramme und Materialien für die Theorieeinheiten werden gesammelt und in den nächsten Jahren Informatiklehrkräften als „Werkzeugkasten“ zugänglich gemacht.

Literaturverzeichnis

[GR13] Göttel, T.; Romeike, R.: Agiler Projektunterricht in der Schulinformatik. In (Breier, N.; Stechert, P.; Wilke, T., Hrsg.): 15. GI Fachtagung Informatik und Schule, Praxisband. Kiel Computer Science 2013/3. Department of Computer Science, CAU Kiel, S. 151–158, 2013.

[Gu08] Gudjons, H.: Handlungsorientiert lehren und lernen: Schüleraktivierung, Selbsttätigkeit, Projektarbeit. 2008.

[HNR07] Hartmann, W.; Näf, M.; Reichert, R.: Informatikunterricht planen und durchführen. Physica-Verlag, 2007.

[IS03] ISB München: , Lehrplan Informatik 10 (NTG), 2003. http://www.isb-gym8-lehrplan.de/contentserv/3.1.neu/g8.de/index.php?StoryID=26435.

[KB09] Kölling, M.; Barnes, D. J.: Java lernen mit BlueJ: Eine Einführung in die objektorientierte Programmierung. Pearson Studium, 2009.

[RG12] Romeike, R.; Göttel, T.: Agile Projects in High School Computing Education: Emphasizing a Learners' Perspective. In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education. WiPSCE '12, ACM, New York, NY, USA, S. 48–57, 2012.

[Ull14] Ullenboom, Ch.: , Java ist auch eine Insel: Das umfassende Handbuch (Galileo Computing), 2014. http://openbook.galileocomputing.de/javainsel11.

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.

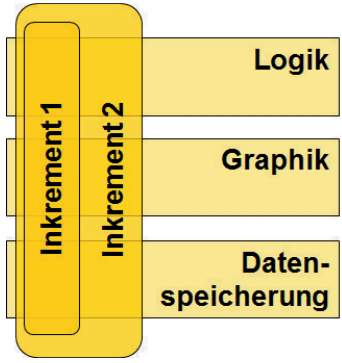

Abb. 1: Inkrementelle Softwareentwicklung über mehrere Schichten

- **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.

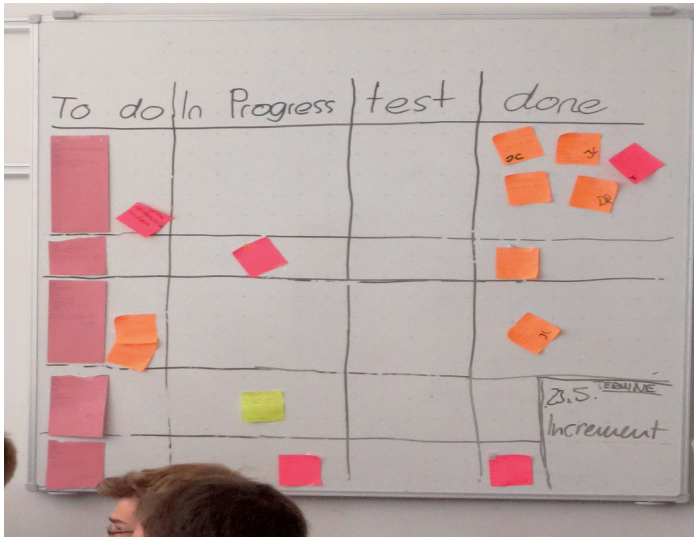

Abb. 2: Taskboard als zentrale Planungsübersicht

- **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).

Agiler Informatikunterricht als Anfangsunterricht

Lennard Kerber[1], Petra Kastl[2] und Ralf Romeike[2]

Abstract: Agile Methoden unterstützen die Organisation und Durchführung von Softwareentwicklungsprojekten und finden inzwischen breite Anwendung im professionellen Bereich. Auch für die Schule sind sie vielversprechend. In diesem Bericht wird beschrieben, wie mit Hilfe adaptierter agiler Methoden Anfangsunterricht in der Programmierung methodisch neu in Form eines „geskripteten Projekts" gestaltet werden kann. In einer ausgearbeiteten Unterrichtssequenz lernen die SchülerInnen selbstreguliert mit dafür erstelltem Material und wenden neu erworbene Wissensbausteine jeweils bei der schrittweisen Weiterentwicklung eines Geschicklichkeitsspiels an. Agile Methoden unterstützen die SchülerInnen dabei unmittelbar und sinnvoll in ihrem Lern- und Arbeitsprozess. Gleichzeitig erwerben die SchülerInnen durch das Einbinden agiler Methoden in die Unterrichtssequenz indirekt bereits eine Methodenkompetenz, die sie in späteren Projekten benötigen. Es werden insbesondere die Anpassungen der agilen Vorgehensweise an den Kontext eines „geskripteten Projekts" und eigene Ergänzungen sowie Beobachtungen und Erfahrungen beschrieben.

Keywords: Agile Methoden, agile Praktiken, Anfangsunterricht, Unterrichtsprojekt

1 Einleitung

Die Verwendung agiler Methoden in professionellen Projekten nimmt seit 2010 erheblich zu [Ko14], denn agile Methoden schaffen Raum für das gestalterische Moment in der Softwareentwicklung, das für manche Projekte sehr wichtig ist [FST13], und wirken sich sehr positiv auf die Motivation der Mitarbeiter und die Kooperation im Team aus [Ko14]. Auch für Schulprojekte sind agile Methoden vielversprechend [RG12]. Im Rahmen des Projekts Agile Methoden im Informatikunterricht (AMI) entwickeln Forscher der FAU Erlangen-Nürnberg zusammen mit engagierten LehrerInnen ein agiles Modell für die Praxis im Informatikunterricht weiter [KR15]. Im Zentrum der Untersuchung stehen Projekte in denen SchülerInnen selbstorganisiert arbeiten und vielfältige fachliche, methodische und soziale Fähigkeiten und Fertigkeiten anwenden und weiterentwickeln – also eigentlich kein Anfängerunterricht. Umso spannender ist das Vorhaben, Anfangsunterricht unter Verwendung der textbasierten Programmiersprache Processing und unter Einbindung sinnvoll gewählter agiler Praktiken methodisch neu zu gestalten, um eine Alternative zu schaffen zu einer langen Lernzeit am Anfang und einer kurzen Projektzeit am Ende, die unter enormem Zeitdruck steht.

Unterricht für Programmieranfänger in einer textbasierten Programmiersprache zu struk-

[1] Otto-Nagel-Gymnasium, Schulstr. 11, 12683 Berlin, l.kerber@otto-nagel-gymnasium.de
[2] Friedrich-Alexander-Universität Erlangen-Nürnberg, Martensstr. 3, 91058 Erlangen, petra.kastl@fau.de, ralf.romeike@fau.de

turieren ist eine Gratwanderung. Stark geführtes, kleinschrittiges Vorgehen mit kurzen praktischen Übungen erzieht zu Unselbständigkeit, schult einseitig die Programmierfertigkeiten und demotiviert leistungsstarke SchülerInnen. Längere Übungsphasen am Rechner dagegen sind schnell ineffektiv und demotivieren Leistungsschwächere, denn sie gehen für viele SchülerInnen mit langem Warten auf Unterstützung einher. Eine Modellierung vorweg ist generell kaum motivierbar, weil ihr Sinn bei kleinen Aufgaben nicht erkennbar ist. Um diese Probleme ein Stück weit aufzulösen, werden beispielsweise Lernaufgaben vorgeschlagen [HNR07] oder das schrittweise Modellieren, Entwickeln und Ausgestalten eines kleinen Spiels über einige Wochen oder Monate hinweg [St09], wobei sich meist Phasen des lehrerzentrierten Unterrichts mit Phasen der Schüleraktivität am Computer abwechseln. Für LehrerInnen bleibt es einer der anstrengendsten Unterrichte.

Dieser Artikel berichtet vom Versuch, Anfangsunterricht methodisch anders zu gestalten. Die Idee war, die SchülerInnen bereits zu einem frühen Zeitpunkt in einem „geskripteten Projekt“ selbstreguliert lernen zu lassen und gezielt einige agilen Praktiken zu nutzen, die die SchülerInnen innerhalb des vorgezeichneten Projekts sinnvoll unterstützen, indem sie beispielsweise einen klaren Organisationsrahmen oder Interaktivität fördernde Handlungsanweisungen vorgeben. Auf der dabei indirekt erworbenen Methodenkompetenz kann im Anschluss in agilen (Softwareentwicklungs-) Projekten aufgebaut werden. Darüber hinaus adressiert das Vorgehen eine Herausforderung der Projektarbeit: Begriffe, Vorgehensweisen und Kommunikationsformen in einem Softwareprojekt müssen in der Regel selbst Gegenstand des Unterrichts werden, ehe sie von den SchülerInnen aktiv angewandt werden können. Das vorliegende Beispiel vermeidet theorielastiges „Lernen auf Vorrat“, indem Terminologie und Aufbau eines Softwareprojekts implizit im Unterricht mit „eingeschliffen“ werden.

2 Rahmenbedingungen

An dem „geskripteten Projekt“ arbeiteten 20 SchülerInnen eines Berliner Wahlpflichtkurses über 12 Wochen jeweils eine Doppelstunde pro Woche. Die Gruppe setzte sich aus verschiedenen 9. Klassen zusammen und es bildeten sich für das Projekt fünf Teams zu je vier SchülerInnen. Für die SchülerInnen war es ihr erster Informatikunterricht. Gemäß internem Curriculum besprachen wir zunächst Algorithmen des Alltags in umgangssprachlicher Formulierung und erarbeiteten dann Kontrollstrukturen mit Robot Karol. Mit Processing[3] setzten sich die SchülerInnen zum ersten Mal im Rahmen des geskripteten Projekts auseinander.

3 „Processing ist eine [...] stark typisierte Programmiersprache [... die] für die Einsatzbereiche Grafik, Simulation und Animation spezialisiert [ist]. Processing hat den Charakter einer stark vereinfachten Version der Programmiersprache Java [...] und richtet sich vorwiegend an Gestalter, Künstler und Programmieranfänger.“

(de.wikipedia.org/wiki/Processing)

Der Computerraum bot mit 14 außen stehenden Computern, von denen 10 genutzt wurden, und einem großen Tisch in der Mitte für Besprechungen im Plenum eine gute Aufteilung sowie genug Raum für Diskussionen in den Teams und ausreichend Abstand zu den anderen Teams.

3 Gestaltung des Lehr-Lernarrangements

Das über die gesamte Einheit des „geskripteten Projekts“ tragende Thema war die Entwicklung eines Geschicklichkeitsspiels, in dem eine Scheibe mit der Maus durch ein 2D-Labyrinth gesteuert wird (vgl. Abb. 1). Wenn die Scheibe zu dicht an die Wände des Labyrinths kommt, wird sie kleiner, die Größe der Scheibe im Ziel bestimmt die erreichte Punktzahl.

Abb. 1: Demoversion der Lehrkraft (links) und unterschiedlich aufwändige Schülerversionen

Nach einer einführenden Doppelstunde, in der ich eine eigene Implementierung des Spiels vorgestellt und unsere agile Vorgehensweise erklärt hatte, waren vier Iterationen, also Zeitfenster fester Länge, geplant. Von diesen vier Iterationen konnten letztendlich aus schulischen Gründen nur drei durchgeführt werden. Jede Iteration war drei Doppelstunden lang und am Ende stellten die SchülerInnen jeweils ihre weiterentwickelten, lauffähigen Prototypen und ein dazu passendes Benutzerhandbuch vor. Jede der drei Iterationen umfasste auf Arbeitsblättern vorgegebene „Student Stories“ und „User Stories“ mit ihren Tasks. Die Student Stories waren vorgegebene Lernaufträge in Form von User Stories (vgl. Abb. 2), die in dieser Iteration erfüllt werden mussten, die User Stories enthielten die in dem gegebenen Zeitfenster zu implementierenden Funktionalitäten des Spiels aus Benutzersicht zusammen mit ihren Tasks, bei deren Umsetzung die neu erlernten Konzepte angewandt wurden.

Inhaltlich war das Lehr-Lernarrangement so gestaltet, dass die SchülerInnen am Ende der ersten Iteration Rechtecke in selbst gewählten Farben zeichnen und damit ihr Spielfeld individuell gestalteten konnten. Sie haben sich dazu mit der *Leinwand* des Processing-Systems und dem Thema *RGB-Farbraum* vertraut gemacht. In der zweiten Iteration wurden die Konzepte *Funktion* (ohne Parameter und ohne Rückgabewert) sowie *Variable* erarbeitet und zur Weiterentwicklung des Spiels verwendet, beispielsweise beim Zeichnen der Rechtecke bzw. der Spielfigur. In dieser Iteration wurde Refactoring, also

die Umstrukturierung von Code ohne Änderung der Funktionalität, genutzt, um bestehenden Code mit Hilfe der neu erlernten Konzepte besser zu gestalten. Am Ende der dritten Iteration schließlich implementierten die SchülerInnen auch *Funktionen mit Rückgabewert* und *Funktionen mit Parametern.* Damit konnte der Spielplan mit einer bewegten Spielfigur, die dem Mauszeiger folgt, programmiert werden. Die Hindernisse waren für die nächste Iteration vorbereitet, in der die Spielfigur bei Berührung verkleinert wird.

Eine Modellierung wurde nicht verlangt, sie wird bei uns erst im zweiten Lernjahr eingeführt, wenn die Beispiele etwas umfangreicher sind. Aber die SchülerInnen mussten Schnittstellen, Variablen und Funktionen nach einem vorgegebenen Schema im Quellcode dokumentieren.

4 Einbindung unterstützender agiler Methoden

Jede Doppelstunde begann mit einem **Stand-Up Meeting**, in dem sich die SchülerInnen jeder Gruppe kurz und knapp klar machen sollten, was sie in der letzten Doppelstunde geschafft haben und was sie für diese Stunde planen. Das heißt, es gab durch mich zwar immer eine Begrüßung, aber die Wiederholung, die Besprechung des Fortschritts und die Sicherung fand individuell in den Gruppen statt und orientierte sich am jeweiligen Stand der einzelnen Gruppe im Lernprozess und im Projekt. Indem die Stand-Up Meetings kurze individuelle Rekapitulationen am Stundenanfang automatisch anstießen, unterstützten sie die Organisation des selbstregulierten Lernens ideal. Hilfreich bei der Rekapitulation waren die Arbeitsblätter der vorangegangenen Stunden und das gruppeneigene **Project Board**, das den Stand der Spieleentwicklung visualisierte. Nach dem Stand-Up Meeting arbeiteten die SchülerInnen jeweils an der Student- bzw. User Story weiter, bei der sie aktuell standen.

Student Stories sind ein von mir für das geskriptete Projekt ergänzend eingeführtes Artefakt. Dabei handelt es sich um Lernaufträge, die von allen SchülerInnen bearbeitet werden mussten. Zu einem Lernauftrag gab es meist eine Lesekarte, auf der neue Theorie stand, zum Teil auch Material, das vom Lehrertisch abgeholt werden musste oder eine kleine Besprechung des Themas mit mir. Bei der Student Story zum Variablenkonzept beispielsweise (vgl. Abb. 2) kamen die Schülergruppen, die so weit waren, zu mir an den Lehrertisch, zu einer kurzen Sitzung. Zur Veranschaulichung des Speicherkonzepts verwendete ich Streichholzschachteln, die den SchülerInnen als Zusatzmaterial auch für die weitere Arbeit an dem Thema zur Verfügung standen. Wenn es zur Theorie praktische Übungen am Computer gab, arbeiteten die SchülerInnen zu zweit als Programming-Pairs zusammen. Beim **Pair Programming**, das den Wissenstransfer unterstützt und planloses Hacken verhindert, ist ein Schüler Driver, er bedient die Tastatur und erklärt, was er sich bei der Programmierung denkt. Der andere Schüler übernimmt die Rolle des Navigators und überlegt sich, ob es eine bessere oder elegantere Lösung gibt. Die Rollen sollten bei uns regelmäßig alle 5 Minuten gewechselt werden.

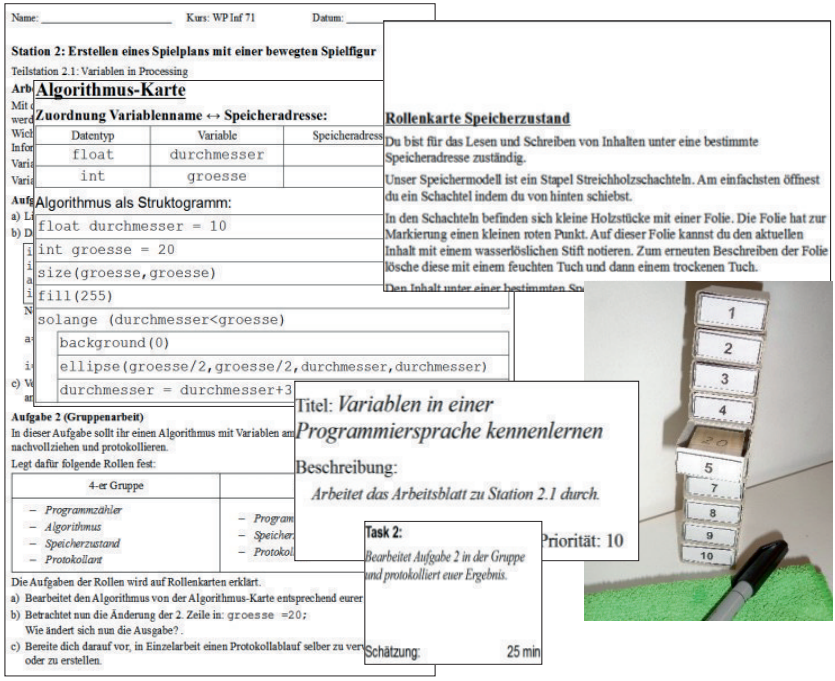

Abb. 2: Student Story zum Thema Variablen

Die **User Stories** dienten der Weiterentwicklung des Geschicklichkeitsspiels unter Verwendung des neu Gelernten. Bei ihrer Implementierung benutzten die SchülerInnen ebenfalls Pair Programming, wobei die beiden Paare eines Vierer-Teams dann teilweise auch an unterschiedlichen User Stories arbeiteten und ihren Code vor dem Test zusammenführten. Zur Koordination und Planung der Spieleentwicklung stand jeder Gruppe ein eigenes **Project Board** zur Verfügung. An dieses waren kleine Zettel mit allen User Stories und den zugehörigen **Tasks** gepinnt, die in den Arbeitsblättern der Iteration enthalten waren. Im Verlauf der Projektarbeit wurden die Tasks nacheinander entsprechend ihrem aktuellen Status in die Spalten „In Progress“ bzw. „Done“ des Project Boards verschoben. Wenn alle Tasks einer User Story erledigt waren und damit die entsprechende Funktionalität im Spiel implementiert war, wurde auch die User Story in die Spalte „Done“ verschoben. Einige User Stories bzw. Tasks stießen **Refactoring**-Tätigkeiten an, die keine neue Funktionalität zum Spiel hinzubrachten sondern dazu dienten, den besehenden Code mit Hilfe neu erlernter Konzepte besser zu strukturieren. Refactoring-Aufgaben wurden ebenfalls von Programming-Pairs durchgeführt und ihr Status am Project Board visualisiert.

Wie weiter oben beschrieben, fanden all diese Tätigkeiten innerhalb von **Iterationen** statt und die Reihenfolge der Bearbeitung war in den Arbeitsblättern vorgegeben. Zur zeitlichen Planung einer Iteration konnten die SchülerInnen das **Planning Poker** nutzen, d. h. für jede anstehende Aufgabe machte jedes Gruppenmitglied zunächst verdeckt eine Zeitschätzung, nach dem Aufdecken mussten stark nach oben oder unten abweichende Schätzungen begründet werden. Am Ende musste sich die Gruppe einigen, beispielsweise auf einen Mittelwert oder auch auf einen Extremwert, bei einsichtiger Begründung.

Am Ende jeder Iteration wurden die **Prototypen,** also die lauffähigen Zwischenstände des Spiels im Plenum besprochen. Jedes Mal stellte dazu ein anderes Gruppenmitglied den jeweiligen Prototypen und das Benutzerhandbuch vor und erläutert sie. In diesen Runden gaben die Mitschüler und ich regelmäßig Feedback, was eine hervorragende Motivation für die weitere Arbeit war. Da wir zur Besprechung immer alle im Plenum versammelt waren und da zu diesem Zeitpunkt stets alle den gleichen Stand hatten, nutzte ich die Runden auch zur Nachgestaltung der Fachsprache oder zur Sicherung zentraler Inhalte. Am Ende jeder Plenumsrunde wurde der Ausblick besprochen. So richteten die SchülerInnen regelmäßig den Blick auf das gesamte Projekt, das Erreichte, das unmittelbar Anstehende und das Ziel, ehe sie in der nächsten Iteration wieder an kleinen Teilschritten arbeiteten.

Die feste, überschaubare Dauer einer Iteration und die sich daran anschließende Prototypenbesprechung gaben dem Lehr-Lernarrangement eine verbindliche zeitliche Taktung und bildeten für die SchülerInnen einen hilfreichen Rahmen zur Orientierung.

5 Erfahrungen und Bewertung

Das agile Vorgehensmodell konnte für eine längere Phase selbstorientierten Lernens bei ProgrammieranfängerInnen gewinnbringend angewandt werden. Die genutzten agilen Praktiken unterstützten die Organisation und Planung, soziales Lernen, die Motivation und das Beurteilen der (Teil-)Produkte optimal. Der selbstregulierte Erwerb fachlicher Kompetenzen im Bereich der Programmierung wurde ergänzt durch die Weiterentwicklung wichtiger sozialer Kompetenzen und ganz nebenbei haben die SchülerInnen eine typische Arbeits- und Vorgehensweise beim Entwickeln von IT-Systemen erlernt und intuitiv reflektiert. Idealerweise folgt dann im selben oder im darauf folgenden Lernjahr ein „richtiges" Unterrichtsprojekt. (Bei mir war es für das Folgejahr geplant, kam aber leider wegen meines Schulwechsels nicht zu Stande.)

Interessant war, dass die SchülerInnen fast alle Praktiken mit Begeisterung aufgenommen haben, ihren Sinn und Nutzen intuitiv reflektierten und sie im späteren Verlauf von sich aus passend bzw. angepasst einsetzten oder wegließen.

Die Project Boards beispielsweise wurden anfangs fortwährend aktualisiert. Vermutlich wegen der Skriptung, der kleinen Gruppengröße, dem geringen Anteil arbeitsteiliger Aufgaben und/oder dem Standort des Boards direkt neben dem Rechner erlebten die

SchülerInnen es mit der Zeit jedoch als unwichtig, dass das Project Board zu jedem Zeitpunkt den aktuellen Stand zeigte. So haben sie später meist erst am Ende einer Doppelstunde die Zettel umgehängt – um zu sehen, was sie geschafft haben und zur Orientierung in der nächsten Stunde. Auch für mich war der Einblick in den Arbeitsstand am Stundenende ausreichend und hilfreich. Die Stand-Up Meetings behielten die SchülerInnen bei, wobei mehr und mehr im Sitzen durchgeführt wurden. Da die Besprechungen trotzdem kurz und zielgerichtet blieben, habe ich es dabei belassen. Refactoring wurde von den Schülerinnen als „Praktik der Profis" akzeptiert und es motivierte den Umbau des Codes, der mit dem schrittweisen Einführen der Lerninhalte verbundenen war. Pair Programming war die einzige Praktik, bei der ich steuernd eingreifen musste, damit der Wechsel der Rollen eingehalten wurde, denn die Beobachtung zeigte, dass sonst meist die Stärkeren programmieren und die Schwächeren zusehen, weil sie sich in der entsprechenden Rolle wohlfühlen. Für mich war es allerdings nicht leicht, konsequent für einen Wechsel zu sorgen. Gleichwohl war es mir wichtig, weil mir diese aus der Praxis stammende Praktik, und insbesondere die Definition der Rollen auch sinnvoll und gewinnbringend erscheint. Das Planning Poker wurde ein, zwei Mal ausprobiert und dann weggelassen, weil es keinen Mehrwert brachte, denn Probleme bei der selbständigen Zeiteinteilung innerhalb einer Iteration gab es nicht. Die Aufgaben einer Iteration waren ja von mir fest vorgegeben und so geplant, dass auch die Schwächeren sie in der gegebenen Zeit bearbeiten konnten. Die Idee bei der Einführung des Planning Poker war auch mehr, dass die SchülerInnen es kennenlernen, um dann selbst zu entscheiden, ob sie es verwenden wollen.

Besonders bereichernd war für mich allerdings das Prototyping, wobei hier verschiedene Aspekte zu meiner sehr positiven Erfahrung beitragen. Beim Vorgehen nach dem Wasserfallmodell liegt erst am Ende ein Produkt vor. Was, wenn eine Gruppe kein lauffähiges Produkt abgibt? Eine schwierige Situation für die Benotung, aber auch unbefriedigend für die Schülergruppe. Durch das Prototyping löste sich diese Situation auf, weil ich früh und regelmäßig Einblick in Zwischenstände hatte und steuernd eingreifen konnte. Darüber hinaus kann ein lauffähiges Zwischenprodukt immer auch benotet werden. In diesem Fall war angekündigt, dass jede Prototypenvorstellung benotet wird. Dazu gab es jeweils klar kommunizierte Anforderungen und Kriterien an den Prototypen und das Benutzerhandbuch und daraus resultierend dann jeweils Teilnoten, die die SchülerInnen einsahen, weil sie transparent und nachvollziehbar zustande kamen. Insbesondere kann aber zu jedem lauffähigen Prototypen Rückmeldung gegeben werden. Ich empfand es als sehr positiv, dass ich so oft die Möglichkeit hatte, zu loben. Auch die SchülerInnen schätzten die Gelegenheit, sich gegenseitig Feedback zu geben. Beides steigerte die Motivation der SchülerInnen sehr und der Ansporn, das Produkt weiter zu entwickeln und erneut zu präsentieren, war bei allen groß. Die SchülerInnen gaben sich wirklich Mühe und es wurden immer tolle Prototypen vorgestellt. Ich konnte beobachten, dass schnelle Gruppen in der verbleibenden Zeit einer Iteration entsprechend der von mir gegebenen Bewertungskriterien mit viel Engagement ihr Spiel ausschmückten und ihr Benutzerhandbuch sehr ansprechend gestalteten. Aber auch schwächere Gruppen kamen mit Fleiß und Einsatz immer zu einem lauffähigen Produkt, für das sie Anerkennung

erfuhren. Insbesondere hat sich hier der Vorteil der agilen Softwareentwicklung bestätigt: Obwohl aus schulischen Gründen auf eine Iteration verzichtet werden musste hatten die SchülerInnen ein Produkt, auf das sie stolz waren und woran sie die Sinnhaftigkeit des neu Erlernten erkannten

Das permanente Loben hat möglicherweise auch dazu geführt, dass ein sonst allgemein auffälliger Schüler bei mir tolle Beiträge lieferte und ein unproblematisches Verhalten zeigte. Positiv überrascht war ich dann aber doch, als eine Kollegin mich fragte, was ich mit meinen Schülern gemacht habe. „Wenn ich ihnen eine Aufgabe gebe“, berichtete sie, „beginnen deine Schüler zu arbeiten, während meine sich erstmal alle melden und Fragen stellen“. Offenbar führten die Projekterfahrungen zu einer nachhaltig selbständigeren Arbeitsweise.

Literaturverzeichnis

[FST13] Fuchs, A.; Stolze, K.; Thomas, O.: Von der klassischen zur agilen Softwareentwicklung. In Praxis der Wirtschaftsinformatik, 2013, 290; S. 17–26.

[HNR07] Hartmann, W.; Näf, M.; Reichert, R.: Informatikunterricht planen und durchführen. Springer, Berlin, 2007.

[Ko14] Komus, A.: Status Quo Agile 2014. Zweite Studie zu Verbreitung und Nutzen agiler Methoden. Ergebnisbericht.

[KR15] Kastl, P.; Romeike, R.: Entwicklung eines agilen Frameworks mit Design Based Research. In (NN Hrsg.): INFOS 2015 - 16. GI Fachtagung Informatik und Schule, 2015.

[RG12] Romeike, R.; Göttel, T.: Agile Projects in High School Computing Education: Emphasizing a Learners’ Perspective. In (Knobelsdorf, M.; Romeike, R. Hrsg.): Proceedings of the 7th Workshop in Primary and Secondary Computing Education. ACM, New York, NY, USA, 2012; S. 48–57.

[St09] Staatsinstitut für Schulqualität und Bildungsforschung ISB Hrsg.: Informatik am naturwissenschaftlich-technologischen Gymnasium Jahrgangsstufe 10. [Erläuterungen und Materialien für Lehrkräfte ; mit CD]. Kastner, Wolnzach, 2009.

Ein Bild vom Wesen der Softwareentwicklung: Erfahrungen aus zwei agilen Projekten

Leonore Dietrich[1], Andreas Gramm[2], Petra Kastl[3] und Ralf Romeike[3]

Abstract: Die beiden Projekte, die hier in getrennten Teilen beschrieben werden, fanden in sehr unterschiedlichen Kontexten statt und zeigen, wie Prozesse unter Verwendung agiler Methoden so an die Bedürfnisse der Beteiligten und die Besonderheiten des Projekts angepasst werden können, dass die intendierten Ziele erreicht werden. Zunächst wird ein „Forschungsprojekt" vorgestellt, das vier Oberstufenschüler mit geringen fachlichen Vorkenntnissen realisierten. Ihre Aufgabe war die Reflexion der erlebten Vorgehensweise. Im zweiten Projekt erlebten Programmieranfänger/innen einer 9. Klasse, wie sie kooperativ selbst Informatiksysteme erschaffen und nach ihren Wünschen gestalten können. Gemeinsam ist beiden Projekten also ihr Fokus auf dem Erlebbar machen eines zeitgemäßen Bilds vom Wesen der Softwareentwicklung.

Keywords: Agile Methoden, Softwareentwicklungsprojekt, Unterrichtsprojekt, Informatikprojekt

1 Einleitung

In diesem Bericht sind zwei Projekte zusammengefasst, die in vielerlei Hinsicht verschieden sind: ein kleines Forschungsprojekt, in dem vier Oberstufenschüler eines Hochbegabtenseminars agile Methoden reflektieren und ein Projekt in einer 9. Klasse, in dem Programmieranfänger/innen mit der visuellen Programmentwicklungsumgebung Scratch ein kleines Computerspiel entwickeln. Gemeinsam ist beiden Projekten, dass sie prozessbezogene Aspekte der Schaffung neuer Informatiksysteme in den Vordergrund stellen und so für Schüler/innen näherungsweise erfahrbar machen, wie Informatiker/innen heute ihren Beruf ausüben. Was aber sind wesentliche Aspekte eines solchen Prozesses, was sind Charakteristika moderner Softwareentwicklung? Die Sicht der Wissenschaft darauf hat sich über die Jahre stetig verändert. Zum Ingenieurswesen und seinem Fokus auf Planung und Problemlösungen ist ergänzend ein kreatives, gestalterisches Moment hinzugekommen und es werden Strategien entwickelt, um angemessen auf Anforderungsänderungen zu reagieren [Me14]. Die Idee von dem einen, idealen Entwicklungsprozess wird abgelöst von der Aufforderung: denke selbst und wähle die guten Ideen, die zu dir und deinem Projekt passen [Ru12, Me14]. Eine Aufforderung, die insbesondere an die Entwickler/innen geht, die den Prozess und ihre Rolle im Prozess mit ausgestalten, regelmäßig reflektieren und optimieren sollen. Hinzu kommt eine verstärk-

[1] Universität Heidelberg, Didaktik der Informatik, Im Neuenheimer Feld 326, 69120 Heidelberg, leonore.dietrich@uni-heidelberg.de
[2] Gymnasium Tiergarten, Altonaer Strasse 26, 10555 Berlin, gramm@gymnasium-tiergarten.de
[3] Friedrich-Alexander-Universität Erlangen-Nürnberg, Didaktik der Informatik, Martensstr. 3, 91058 Erlangen, petra.kastl@fau.de, ralf.romeike@fau.de

te Betonung von Eigenverantwortung und Teamorientierung in modernen Entwicklungsprozessen. Die hier beschriebenen Schulprojekte zeigen, dass dieses Wesen von Softwareentwicklung auch in Schulprojekten motivierend und realistisch erlebbar gemacht werden kann. Sie geben Antwort auf die Frage, welches Bild der Informatik, eines Softwareentwicklungsprozesses sowie der Anforderungen von Projektarbeit an Entwickler/innen wir Jugendlichen in einem „agilen Projekt" vermitteln können und wie Schüler/innen ihr Erleben reflektieren.

2 Teil 1: Agile Methoden – nur cool oder mehr? Ein Projekt zur Reflexion des Prozesses durch die Schüler

Das Projekt „Agile Methoden in der Softwareentwicklung" fand im Rahmen des Hector-Seminars statt, einem Baden-Württembergischen Programm zur langfristigen Hochbegabtenförderung, das in der Oberstufe mit einer einjährige Kooperationsphase abschließt, in der Teams, meist in Zusammenarbeit mit einer Hochschule, ein kleines Forschungsprojekt durchführen. Vier Schüler, aus Mannheim und Heidelberg, haben in dieser Phase ein Softwareprojekt mit einer agilen Vorgehensweise [RG12] realisiert. Ziel war, das Vorgehen und die agilen Methoden zu erleben, zu reflektieren und abschließend vorzustellen. Die Projektumsetzung begann im Januar und lief bis Ende des Schuljahres. Im Mittel fand in dieser Zeit ein begleitetes, vierstündiges Treffen pro Monat an der Universität statt, parallel dazu gab es selbständige Treffen. Die fachlichen Vorkenntnisse der Schüler waren überwiegend gering. Sie hatten teils in der Mittelstufe einzelne Kurse zur Robotik (Lego Mindstorms) und zum Einstieg in die Programmierung mit Java (Greenfoot Kara) belegt, aber kaum tiefere, objektorientierte Konzepte verinnerlicht. Ein Schüler hatte grundlegende Java-Kenntnisse, Prozessmodelle aus der Softwareentwicklungstechnik waren nicht bekannt. Allerdings sind die Schüler sehr reflektiert, verfügen über eine extrem schnelle Auffassungsgabe und haben eine außergewöhnlich ausgeprägte Fähigkeit zu analytischem und strukturiertem Denken sowie eine hohe Abstraktionsfähigkeit.

2.1 Der Projektverlauf

Das Spiel, welches mit Greenfoot im Projekt entwickelt wurde, wurde primär als Produkt der Anwendung der ausgewählten Methoden und Strategien thematisiert. Deshalb wurde im ersten Treffen zunächst erläutert, wie Softwareentwicklungsprojekte mit agilen Vorgehensweisen ablaufen. Ausgewählte Videos unterstützten die Ausführungen und vermittelten einen Einblick in das Vorgehen bei professionellen Projekten. Anschließend wurde besprochen, wie es im konkreten Projekt aussehen kann. Dabei sind wir auf folgende Praktiken und Artefakte eingegangen: Stand-Up-Meetings[4], User Stories und

[4] Treffen des Teams, um sich genseitig über die jeweiligen Aktivitäten in arbeitsteiligen Phasen zu informieren.

Tasks[5] sowie ihre Aufwandsabschätzung mit Hilfe des Planning Pokers[6], Sprints[7], Pair Programming mit den Rollen des Drivers und des Navigators, Refactoring[8], Prototypen, die Retrospektive und das Project Board als Planungs- und Informationsbereich.

Im konkreten Kontext hätte sich ein elektronisches Project Board angeboten, da die Treffen an verschiedenen Orten stattfanden und die Schüler von unterschiedlichen Schulen kamen. Allerdings wollte ich den Schülern die Möglichkeit bieten, nach getaner Arbeit für alle sichtbar und haptisch erlebbar einen Zettel umzuhängen. Deshalb bestand unser Project Board aus einem für den Transport gefalteten und zu den Treffen mitgebrachten Plakat (vgl. Abb. 1). Die vierstündigen betreuten Treffen umfassten zwei Sprints von je 45 Minuten. Das Stand-Up-Meeting zu Beginn diente dem Rekapitulieren des vorangegangenen Treffens und ging dann in die Planung des nächsten Sprints über. Während der Planungs- und Entwurfsphase wurde vor allem viel diskutiert. Dokumentiert wurden ein Klassendiagramm, das im Rahmen eines Refactorings erstellt wurde und einige Abläufe in Form von Struktogrammen (vgl. Abb. 1). Für das Projekt und seine Ziele war diese Form der Planung passend, denn die Schüler hatten wenig Vorkenntnisse im Bereich von Spezifikationssprachen und das Produkt war bezogen auf die Fähigkeiten der Schüler zu strukturierendem und abstrahierendem Denken wenig komplex. Außerdem wurde so die "besondere „Schülerklientel" zum Kommunizieren gebracht und es entstand viel Gelegenheit zum Wissenstransfer. Die Reflexionen der Sprints am Ende der Treffen hingegen wurden gut dokumentiert und die Schüler haben Schlussfolgerungen für ihren Entwicklungsprozess schriftlich festgehalten.

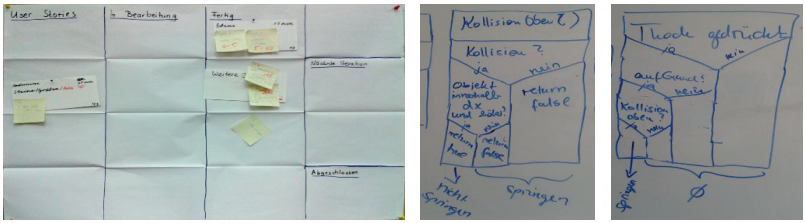

Abb. 1: Project Board und Struktogramme

2.2 Beobachtung und Bewertung

Aus Lehrersicht[9]:. Problematisch war die Teamgröße von nur 4 Schülern. Wenn einer der vier Schüler kurzfristig ein Treffen absagte, waren sinnvolle Sprints kaum möglich. Eine Herausforderung stellte das agile Vorgehen bezüglich der Erwartungen an kommu-

[5] Funktionalitäten aus Kundensicht sowie Aufgaben, die zur Umsetzung aus Entwicklersicht notwendig sind.
[6] Praktik zum gemeinsamen und spielerischen Abschätzen des Arbeitsaufwands.
[7] Feste Zeitfenster, in denen der Prototyp inkrementell in Miniprojekten weiterentwickelt wird: auch Iteration.
[8] Das Umstrukturieren des Quelltextes ohne Änderung der Funktionalität.
[9] Hier werden nur die Schülersicht ergänzende Aspekte beschrieben, abweichende gab es nicht.

nikative und kooperative Fähigkeiten dar – zumindest für drei der vier Schüler. Beim ersten Stand-Up-Meeting beispielsweise standen sie da wie „bestellt und nicht abgeholt". Im Verlauf wurde die Kommunikation untereinander beobachtbar besser: nach drei Sprints war beispielsweise zu erkennen, dass die Diskussionen beim Planen und später beim Abschätzen des Aufwands für die Tasks wesentlich differenzierter wurden und die Beteiligung gleichmäßiger verteilt war. Unterstützt wurde die Entwicklung vermutlich auch durch eine wechselnde Zusammenstellung der Pairs beim Programmieren, die für den Wissenstransfer förderlich war. Insgesamt hatte ich den Eindruck, dass sie durch die agile Vorgehensweise und die sogenannten Zeremonien einen für sie praktikablen Fahrplan hatten und eine Struktur, in der sie Fehler erkennen und Probleme lösen konnten. Sie sind planvoll vorgegangen, ohne dass ich als Lehrerin immer den nächsten Schritt vorgeben und extra motivieren musste. Das Planning Poker z. B. traf spontan auf große Zustimmung, trotz anfänglicher Schwierigkeiten aufgrund geringer Programmiererfahrung. Die Schüler reflektierten die erlebten Abweichungen nach jedem Sprint und bezogen die Erfahrungen in die folgenden Abschätzungen ein: „Das denke ich ist etwas / viel mehr / weniger aufwändig als [...], weil [...]". Dabei fielen ihnen auch Aspekte auf, die sie beim Planen der Tasks zu ungenau besprochen hatten, weil dann die Schätzungen auseinander lagen, d. h. die Abschätzung hatte auch eine Kontrollfunktion. Erstaunt hat mich, dass die Schüler von sich aus Fehler, die sie im Test gefunden haben, strukturierten und klassifizierten, um diese dann bei den zukünftigen Prototypen für Tests wieder zu verwenden.

Insgesamt konnte ich viel zuhören und in den Diskussionen von der Planung bis zur Retrospektive verfolgen, wie die Schüler denken, wie sie an Problemstellungen herangehen, sich Unbekanntes erschließen und wie sich ihre Fähigkeiten und Fertigkeiten im Laufe des Projekts entwickeln. Der Aufbau agiler Entwicklung erwies sich für die Schüler als anregend, die Bezeichnungen der Zeremonien sind nah an der Schülersprache und Elemente wie Planning Poker integrieren eine Art Gamification-Ansatz in die Projektsteuerung. Über diese Elemente lassen sich die Schüler gut abholen. Für das nächste Projekt wird die Gruppe allerdings größer ausfallen, um auch bei Krankheit eines Teammitglieds wenigstens zwei Pairs für eine weitere Iteration zusammenstellen zu können. Eine Teamgröße von 6-7 Schülern halte ich auf der Basis der Projekterfahrung für sinnvoll. Interessant wäre auch die Erprobung einer grafischen Programmierumgebung, die den Fokus noch mehr weg von Implementierungsproblemen und hin zum agilen Prozess verschiebt.

Aus Schülersicht: Der erste Durchlauf eines Sprints war für uns eine neue Erfahrung. Die kurzen Phasen erforderten gezieltes Arbeiten am Projekt und damit auch klare Kommunikation und Rollenverteilung. Die Formulierung von User Stories schien uns zunächst schnell von der Hand zu gehen. Bei der Formulierung des Tasks wurde dann aber klar, dass die Stories zum Teil zu klein waren – nur eine Task enthielten – und zum Teil viel zu umfassend. Das anschließende Planning Poker brachte weitere Schwierigkeiten und einige Diskussionen um Zeit- und Implementierungsaufwand. Im ersten Sprint stellte sich dann auch heraus, dass wir uns in einigen Punkten verschätzt hatten, wobei insbesondere die zu großen User Stories dann im vorgegebenen Zeitfenster nicht um-

setzbar waren. Diese wurden in der Sprint Retrospektive angesprochen und für den nächsten Sprint wieder aufgenommen. Die zu Beginn des zweiten Sprints zunächst vorgenommene Neuformulierung brach nun die zu groß geratenen User Stories in kleinere auf. Bei dieser Aufgabe holten wir uns bisweilen auch Feedback bei der Lehrerin. Ein wirklicher Kunde oder eine entsprechende Rolle – diese wurde in unserem Fall dann von der betreuenden Lehrerin übernommen – sind für die Entwicklung sehr hilfreich.

Im Rahmen der weiteren Sprints konnten wir den Umgang mit den Zeremonien verinnerlichen und uns in die schnelle Entwicklung in kurzen Iterationen einarbeiten. Als motivierend wirkte das stets lauffähige Spiel, die Möglichkeit, jederzeit zu testen und, die Auswirkungen der eigenen Programmierung direkt zu sehen. Schwierigkeiten hatten wir beim unbetreuten Arbeiten insofern, als dass wir die engen Zeitvorgaben nicht konsequent einhielten – hier ist wohl ein Projektmanager hilfreich wie ihn die betreuende Lehrerin bei den Präsenzterminen darstellte. Gleiches gilt für den Rollenwechsel in den Pairs innerhalb der Implementierungsphasen. Bei weiteren Präsenzterminen galt deshalb dem Zeitplan und dem Pair-Wechsel besondere Aufmerksamkeit. Wir konnten durch den Pair-Wechsel auch von dem Teammitglied lernen, das die besten Programmierkenntnisse mitbrachte.

Nach drei Iterationen kamen wir an eine Stelle, an der wir einige Programmierprobleme nicht mehr selbständig lösen konnten und fachliche Hilfe benötigten. Der Seminarleiter zeigte uns verschiedene Möglichkeiten zur Lösung des Problems auf. Bei der Umsetzung stellte sich jedoch heraus, dass wir gleichartige Funktionalitäten an vielen verschiedenen Stellen brauchten und somit eine Überarbeitung unserer Gesamtstruktur nötig wurde. Hierfür unterbrachen wir das agile Szenario[10] und entwickelten mit Unterstützung durch die Lehrerin ein Klassendiagramm aus Metaplankarten und Post Its. Das Experimentieren mit den Karten, aus denen wir das Klassendiagramm zusammenbauten, war einfach und schnell und man konnte gut sehen, wie die einzelnen Programmteile zusammenhängen. Indem wir Klassen und Methoden verschoben und verschiedene Strukturen diskutierten, fanden wir eine sinnvolle Struktur. Die Umsetzung des Refactorings übernahm ein Pair, während das andere sich mit einem weiteren identifizierten Problem befasste: der einheitlichen und standardkonformen Benennung von Klassen und Methoden. Als relativ einfach nahmen wir das Zusammenführen des Codes wahr, wobei die zwei Pairs beim Implementieren versuchten, jeweils in verschiedenen Klassen zu arbeiten. So mussten nur die einzelnen Klassen zusammenkopiert werden. Insgesamt war es spannend, den agilen Ansatz kennenzulernen. Bessere Vorkenntnisse in der Programmiersprache und der -umgebung hätten uns allerdings den Einstieg sicher erleichtert.

[10] In einer Iteration wurde kein Produktinkrement erstellt sondern ein Refactoring geplant und umgesetzt.

3 Teil 2: Das Henne-Ei Problem – Wie Programmieranfänger/innen ein Projekt stemmen können

Eine zentrale Frage ist, was wir mit Projekten im Informatikunterricht erreichen wollen. Grundsätzlich finde ich es schwierig, im gegebenen zeitlichen Rahmen Vorgehensweisen bei der Entwicklung größerer Softwaresysteme für Lernende erlebbar zu machen und sie über die Phasen hinweg zu motivieren. Ein typisches Problem ist, dass sie gar keine Analyse machen können, bevor sie nicht ein Projekt durchlaufen haben und wissen, was die eine oder andere Modellierungsentscheidung später bewirkt, was es bedeutet, etwas als Klasse, Interface oder Entität zu modellieren. Ohne ein ordentliches Modell kann ich kein sinnvoll strukturiertes Produkt erstellen und ohne einmal ein Produkt gesehen zu haben, kann ich nicht sinnvoll modellieren – ein Henne-Ei-Problem. Unzufrieden bin ich auch damit, dass Tests aus Zeitgründen zurücktreten müssen und für eine Reflexion trotzdem viel zu wenig Zeit bleibt. Bis aber die Schüler/innen überhaupt über die Voraussetzungen verfügen, um linear verlaufende Projekte angehen zu können, ist oft schon bei vielen ihr Interesse für das Fach der Auffassung gewichen, dass Informatik viel zu schwer ist.

Wichtig ist mir deshalb, ihnen am Anfang zu zeigen, was man mit Informatik machen kann und wie große Informatiksysteme entstehen. Ich möchte ihnen zeigen, dass sie etwas schaffen können, sie für das Fach begeistern und ihnen ein inspirierendes Bild vom Beruf eines Informatikers / einer Informatikerin vermitteln. Gerade im Anfangsunterricht möchte ich erlebbar machen, dass es in diesem Beruf um Gestalten und um Teamwork geht und dass man sich in der Informatik gut organisieren und zusammen tolle Sachen machen kann. Deshalb finde ich ein iteratives Vorgehen sehr verlockend. Man hat kurze Zyklen, sodass die Schüler/innen in zwei Wochen in einem Miniprojekt alles kennenlernen und dabei ein interessantes Zahnrädchen einer größeren Lösung entwickeln. In der nächsten Iteration kommt ein weiteres Zahnrädchen dazu. Es ist mehr als die Aneinanderreihung von Umsetzungen kleiner, unabhängiger Probleme, denn es fügt sich zu etwas Größerem zusammen. Wenn daran sechs Leute in Kooperation arbeiten, entsteht schnell etwas Spannendes. Das Endprodukt muss nicht perfekt sein, es muss nicht jede Idee umgesetzt sein. Trotzdem haben die Schüler/innen einen lauffähigen Prototypen, den sie selbst gemeinsam entwickelt haben und sie wissen, dass und wie es weitergehen könnte. Auch der Test verändert sich dadurch, dass er an einem inkrementell wachsenden, realen Produkt stattfinden kann, nicht auf einer Kommandozeilenebene weit weg vom eigentlichen Produkt.

3.1 Rahmenbedingungen

Das Projekt fand in einer 9. Klasse in einem Wahlpflichtkurs mit einer Doppelstunde pro Woche statt. Die Gruppe bestand aus 20 Schüler/innen, die zum größten Teil nichtdeutscher Herkunftssprache waren, weshalb sich die Kommunikation in kooperativen Arbeitsformen auch immer zur Förderung der Sprachkompetenz im Sinne einer Sprachbil-

dung im Fachunterricht anbietet. Für die Schüler/innen war es ihr erstes Schuljahr mit Informatikunterricht. Das Projekt wurde in der frühen Phase des Kurses mit Scratch durchgeführt. Aufgabe war es, exemplarische Komponenten eines Jump-and-Run-Spiels zu entwickeln. Das Projekt musste umständehalber zwei Mal unterbrochen werden. Die Vorübungen fanden im Herbst statt, die Formulierung der User Stories im Januar und die Realisierung im Frühjahr.

3.2 Einbettung in den Unterricht und Anpassungen an das konkrete Projekt

Vor dem Projekt wurden Kollisionen und Variablen in Scratch behandelt. Die Schüler/innen konnten Zustände und Zustandsveränderungen speichern und wussten, wie eine Kollision als Ereignis erfasst werden kann. Auf dazugehörige Arbeitsblätter mit erklärenden Beispielen konnten sie im Projekt zurückgreifen. Außerdem wurde vorbereitend auf das Projekt geübt, wie einzelne Objekte in Scratch exportiert und importiert werden, wobei deutlich wurde, dass eine Codeintegration relativ einfach abläuft, solange man nicht in gleichen Sprites arbeitet. Diese Übung haben die Schüler/innen als Pair Programming durchgeführt und damit diese Praktik bereits vorbereitend geübt.

Die agile Vorgehensweise sollte eine effektive Produktentwicklung mit Scratch unterstützen und wurde entsprechend angepasst: Da die User Stories sehr klein waren und nur einige wenige Aufgaben umfassten, bedurfte es in Verbindung mit Scratch keiner zusätzlichen Formulierung von Teilaufgaben aus Entwicklersicht. Deshalb wurde nicht zwischen User Stories und Tasks unterschieden. Die User Stories wurden notiert und am Project Board befestigt, ihr Aufwand wurde mit ein bis drei Sternchen[11] und ohne Planning Poker abgeschätzt (vgl. Abb. 2). Eine logische Bearbeitungsreihenfolge ergab sich im Verlauf von selbst, d. h. auch auf eine Priorisierung konnte verzichtet werden, wodurch die Vorarbeiten weiter verkürzt wurden. Stand-Up Meetings fanden zu Beginn jeder Doppelstunde statt und gaben den Lernenden eine Struktur vor, in der sie selbstständig den Stand ihrer Arbeit rekapitulieren und anschließend die Doppelstunde planen konnten, wobei die wesentliche Aufgabe der Planung in der Auswahl geeigneter User Stories bestand. Während der Stunde konnten in der Gruppe auftretende Fragen und Probleme auch in einem Stand-Up-Meeting vor dem Board besprochen werden (vgl. Abb. 2). Da es keinen festgelegten Funktionsumfang gab, der im gegebenen Zeitrahmen umgesetzt werden musste, genügte das Board, um den Projektfortschritt zu verfolgen, auf ein Burn Down Chart zur graphischen Darstellung konnte verzichtet werden. Die konkrete Umsetzung mit Scratch erfolgte im Pair Programming, wobei die Rolle des Navigators aufgrund der wenigen Konzepte und Datenstrukturen, die bekannt waren, eingeschränkt war. Wichtig war, dass der Driver stets seine Ideen bei der Umsetzung ausdrückte und die Schüler/innen so lernten, ihre Programmierung zu beschreiben bzw. kritisch zu hinterfragen.

Es wurden zwei Prototypen entwickelt, von denen nur der zweite zusammen mit einer

[11] Wenig, mittel bzw. viel Aufwand.

Reflexion über das im Projekt Gelernte im Plenum vorgestellt wurde, um eine längere Phase der konzentrierten Projektarbeit zu haben, nachdem das Projekt im Vorfeld zwei Mal unterbrochen worden war. Ihre individuellen Leistungen haben die Schüler/innen abschließend in den Gruppen besprochen und bewertet.

3.3 Die Projektdurchführung

Als Einstieg wurde das Ball Point Game [Gl15] mit allen 20 Schülerinnen und Schülern gemeinsam gespielt (vgl. Abb. 2). Sie hatten einen Heidenspaß und eine deutlich sichtbare Optimierung. Es war eine gelungene Motivation, die zeigte, wie wichtig Absprachen für eine erfolgreiche Kooperation sind und wie man einen iterativen Prozess durch stringentes Planen, Handeln und Reflektieren optimieren kann. In der verbleibenden Zeit dieser Doppelstunde wurde der Kurs in drei Gruppen geteilt, welche erste Ideen für ihr Jump-and-Run-Spiel sammelten, diskutierten und als User Stories formulierten. In einer weiteren Doppelstunde wurden diese User Stories konkretisiert und ergänzt. Die Umsetzung erfolgte in zwei Iterationen von jeweils zwei Doppelstunden an deren Ende jeweils ein Prototyp vorliegen sollte. Teilweise wurden in den Planungen weitere User Stories ergänzt, soweit dies die Umsetzung der Spielidee erforderlich machte. Wenn sie wollten, konnten die Schüler/innen sich ihr Projekt auf einen Stick kopieren und daran zu Hause weiterarbeiten. Die dabei umgesetzten Funktionalitäten wurden in der folgenden Doppelstunde der Gruppe vorgestellt und in das Projekt integriert. Zwei oder drei Mal sind in dieser Phase Probleme aufgetreten, die entweder alle hatten oder auf die sie in Kürze stoßen würden. In diesen Situationen habe ich das Thema zu Beginn der folgenden Doppelstunde kurz im Plenum aufgegriffen und von einzelnen Gruppen erarbeitete Lösungen für das Problem dem gesamten Kurs vorstellen lassen.

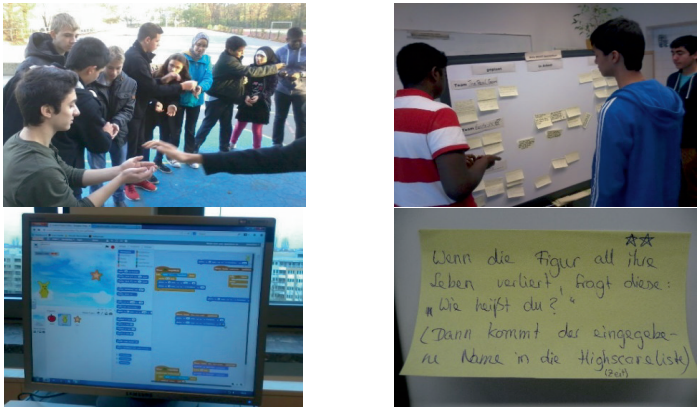

Abb. 2: Ball Point Game, am Project Board, Implementierung, User Story

In der abschließenden Präsentation sollten die Schüler/innen die Umsetzung einzelner Funktionalitäten erläutern können und jede Gruppe sollte eine Zusammenfassung ihrer

individuellen Reflexionen zur der Frage „Was habe ich im Projekt gelernt?“ vorstellen.

Eine Herausforderung stellt die Bewertung von Leistungen dar, die von der Gruppe bzw. individuell in einem Projekt erbracht werden, da zwei unterschiedliche Interessen kollidieren: Zum einen will ich sie befähigen, etwas zu tun und gleichzeitig messe ich, was sie tun. Ein weiterer Punkt ist die Frage, welchen Wert eine individuelle Idee hat und welchen Wert die Fähigkeit der Gruppe, sie erfolgreich umzusetzen. Nachdem die Schüler/innen ihre Projekte selbstorganisiert durchgeführt haben, war es mir wichtig, sie in den Prozess der Bewertung einzubeziehen. Gemäß dem Prinzip einer Poolnote bekam jede Gruppe von mir entsprechend ihrer Gesamtleistung eine Anzahl von Punkten. Die Aufgabe der Gruppe bestand nun darin, sich auf eine Verteilung der Punkte auf die Gruppenmitglieder zu einigen und sich so mit der Frage auseinandersetzen, welche Fähigkeiten und Fertigkeiten jedes einzelnen Mitgliedes die Gruppe im Projekt besonders gut vorangebracht haben und welche eher hemmend waren. Die so erzielte Note umfasste sowohl die Leistung bezüglich des Produkts als auch den Beitrag zur Zusammenarbeit.

3.4 Beobachtungen und Erfahrungen

Die Entscheidung, das Projekt früh im Schuljahr durchzuführen war gut, weil es bei den Lernenden eine positive Einstellung zum Fach erzeugte. Auch das Thema würde ich so wiederwählen, zum einen waren sie von Anfang an interessiert und konnten ihre Erfahrung mit Spielen einbringen, zum anderen entstand im Laufe der Zeit eine Art Wettbewerbssituation. Sie interessierten sich mehr und mehr für die Produkte der anderen. Diese Wettbewerbssituation wirkte zusätzlich anregend, ebenso wie das Peerfeedback. Motivation und Engagement führten dazu, dass schnell gute Prototypen da waren. Die positive Bestärkung der Selbstwirksamkeit durch die iterative Vorgehensweise und die Prototypen sehe ich sehr positiv. Und da sich der Ablauf wiederholt und eine klare Struktur hat, musste ich nicht wie früher moderieren und den nächsten Schritt vorgeben. Die Schüler/innen arbeiteten eigenständiger als in Projekten nach dem Wasserfallmodell und zunehmend selbstbewusster. Mir hat gefehlt, dass sie während des Projekts in fachlicher Hinsicht nicht nur bekannte Fähigkeiten ausbauen und vertiefen, sondern sich im Rahmen des Projekts auch neue Konzepte erarbeiten. Im nächsten Projekt werde ich daher Anregungen bereitstellen, die die Erarbeitung neuer Fachkonzepte initiieren.

Für mich sehr entlastend war, dass von Anfang an klar kommuniziert war: Wir werden nur ein Stück des Spiels implementieren, es wird funktionieren, aber nicht ausgereift sein. Es bestanden weder Anspruch noch Notwendigkeit, in der gegebenen Zeit alle Funktionalitäten umsetzen zu müssen, um ein sinnvolles Produkt zu haben. Darüber, wie sich das bei Oberstufenprojekten umsetzen lässt, muss ich noch nachdenken, hier jedenfalls war ich entspannt und die Schüler/innen waren mit ihren Ergebnissen zufrieden.

Verantwortlich für ihr Projekt haben sich fast alle Schüler/innen gefühlt. Einzelne, die sich herausnehmen, gibt es wahrscheinlich immer, aber hier ist ein solches Verhalten

deutlich aufgefallen und es war für diejenigen unangenehmer als gewöhnlich. In einer Gruppe hat es mit der Teamarbeit nicht gut funktioniert. Auch dort haben sich alle bemüht, etwas beizutragen. Allerdings war in diesem Team eine sehr leistungsstarke Schülerin mit einer schnellen Auffassungsgabe. Sie fand schnell Lösungen und setzte sie um. Leider hat es die Gruppe aber nicht geschafft, dieses Wissen zu transferieren und gemeinsam zu nutzen, obwohl ich dies wiederholt angeregt habe. Diese Schülerin hatte in der Gruppe keinen leichten Stand. Ich vermute, dass sich die anderen Gruppenmitglieder nicht von ihr unterstützt fühlten bzw. es als besondere Herausforderung empfanden, neben ihr auch gute Leistungen zeigen zu können.

Bei der Bewertung haben sich zwei Gruppen sehr schnell geeinigt, die Punkte gleich aufzuteilen. An dieser Stelle war es mir nicht wichtig, einzugreifen oder mir die Überlegungen darlegen zu lassen, da sich meiner Beobachtung nach alle für ein gutes Arbeitsergebnis engagiert hatten. Interessant war wiederum die Gruppe, in der es Spannungen gegeben hatte. In dieser Gruppe gab es entsprechend große Diskussionen: Die leistungsstärkere Schülerin meinte, dass ihr Beitrag mehr Anerkennung finden müsste, die anderen Gruppenmitglieder betonten, dass erst die Tatsache, dass sie der Gruppe eine Überarbeitung zugesagt, aber nicht eingehalten hat, ein besseres Gruppenergebnis verhindert hat. Nach langer Diskussion mit wenig Verständnis für die jeweils andere Position, haben sie sich als Kompromiss auf die gleiche Punktzahl für alle Gruppenmitglieder geeignet. Die Schüler/innen haben damit eine sehr ambivalente Situation bewältigt, die ihnen auch im späteren Leben begegnen wird. Am Ende formulierten sie dann auch in der Reflexion: Absprachen treffen ist wichtig, aber mühsam und teilweise schwierig. Aber ohne Absprachen kann eine fruchtbare Zusammenarbeit nicht funktionieren. Neben dieser Erfahrung war die Präsentation der Arbeitsergebnisse aber vor allem von Stolz auf das Erreichte geprägt. Auch wenn es sich dabei nur um kleine Funktionen handelte: Die Schüler/innen haben erfahren, dass sie gemeinsam selbst Informatiksysteme erschaffen und nach ihren eigenen Wünschen und Interessen gestalten können und waren nun hoch motiviert, weitere Themengebiete der Informatik zu erarbeiten.

Literaturverzeichnis

[Gl15] Glogler B.: Ball Point Game. http://borisgloger.com/scrum/materialien/tools/, 24.04.2015.

[Me14] Meyer, B.: Agile! The good, the hype and the ugly. Springer, 2014.

[RG12] Romeike, R.; Göttel, T.: Agile Projects in High School Computing Education: Emphasizing a Learners' Perspective. In (Knobelsdorf, M.; Romeike, R. Hrsg.): Proceedings of the 7th Workshop in Primary and Secondary Computing Education. ACM, New York, NY, USA, 2012; S. 48–57.

[Ru12] Rumpe, B.: Agile Modellierung mit UML. Codegenerierung, Testfälle, Refactoring. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

Repositories zur Unterstützung von kollaborativen Arbeiten in Softwareprojekten

Peter Brichzin[1] und Thomas Rau[2]

Abstract: In der professionellen Softwareentwicklung ist die Verwendung eines Repositoriums mit Versionskontrolle ein Standardwerkzeug, um verteilt an gemeinsamen Projekten zu arbeiten. Auch bei Softwareprojekten im Informatikunterricht der Oberstufe unterstützt der Einsatz kollaboratives Arbeiten, etwa beim regelmäßigen Zusammenfügen (zu Prototypen), bei der Verfügbarkeit des Quelltexts auch bei abwesenden Teammitgliedern und bei der Dokumentation der Arbeit. In diesem Aufsatz werden praktische Unterrichtsszenarien mit der Entwicklungsumgebung BlueJ und dem dort integrierten svn-Client exemplarisch vorgestellt.

Keywords: Kollaboratives Arbeiten, Subversion, Softwareentwicklung, Projekt, Java, BlueJ, agil

1 Über die Notwendigkeit von Versionskontroll-Systemen

1.1 Nöte in der Dateiverwaltung bei der Softwareentwicklung

In der bekannten Webcomic-Serie xkcd gibt es einen Cartoon, der sich über die Namensgebung in Dateiverzeichnissen lustig macht (s. Abb. 1). Ähnlich sieht das auch in den Quelltextordnern von Schülerinnen und Schülern aus – dort heißen die gespeicherten Dateien beispielsweise „Game1“, „Game1 (Kopie)“ und „Game2 (funktioniert)“. Es ist verständlich und nützlich, Sicherheitskopien im Laufe der Softwareentwicklung anzulegen. Aber spätestens dann, wenn man mit anderen Leuten gemeinsam an einem Projekt arbeitet, wird die Vielzahl der Projektversionen und Namenskonventionen unübersichtlich, und wenn man gar nach einer längeren Unterbrechung wie den Ferien ein begonnenes Projekt fortsetzen möchte, gleicht es einer Art Ratespiel, die letzte funktionierende Fassung herauszufinden, um daran weiterzuarbeiten.

Es gibt noch weitere Schwierigkeiten bei der Verwaltung von Quelltexten bei Oberstufen-Softwareprojekten: Wie lassen sich die Teilergebnisse regelmäßig zusammenfügen, sodass nicht erst gegen Ende des Projektzeitraums Defizite an Schnittstellen aufgedeckt werden? Wie vermeidet man, dass das Team in einzelnen Unterrichtsstunden blockiert ist, weil der Schüler mit wesentlichen Teilen des aktuellen Quelltextes krank zu Hause ist? Wie ermöglicht man, dass begeisterte Schülerinnen und Schüler zu Hause weiterentwickeln können und damit dem Projektfortschritt einen wesentlichen Impuls geben? Wie lässt sich einfach dokumentieren, wer, wann, welche Teile der Software bearbeitet

[1] LMU München, Institut für Informatik, Oettingenstr. 67, 80538 München, brichzin@tcs.ifi.lmu.de
[2] LMU München, Institut für Informatik, Oettingenstr. 67, 80538 München, thomas.rau@ifi.lmu.de

hat, sodass man sowohl jederzeit einen Überblick über den aktuellen Stand hat, als auch rückwirkend eine Übersicht über das Engagement Einzelner für das Projekt hat?

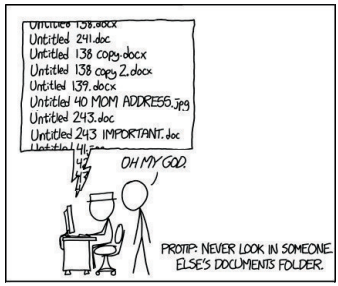

Abb. 1: „Never look in someone else's documents folder", https://xkcd.com/1459/

1.2 Versionskontroll-Systeme als Lösung

In der professionellen Softwareentwicklung in Unternehmen und im Open-Source-Bereich ist der Einsatz von Versionskontroll-Systemen die Antwort auf die oben gestellten Fragen. Zu den Funktionalitäten dieses Werkzeuges gehören:

- Verteilter Zugriff auf Dateien als Voraussetzung für kollaboratives Arbeiten
- Versionierung: Übersicht über verschiedene Versionen inklusiv der Dokumentation wer, wann, was geändert hat
- Datensicherheit, u.a. durch das Rücksetzen der Datei auf eine ältere Version
- Automatisches Zusammenführen (engl. merge) von Quelltexten bzw. Konflikterkennung, falls mehrere Entwickler Veränderungen im selben Bereich durchführen

Bekannte Versionskontroll-Systeme sind u.a. Apache Subversion (seit 2000), Git (seit 2005) und Mercurial (seit 2005). In der Schule werden diese Lösungen bisher selten eingesetzt. Gründe dafür sind möglicherweise, dass diese Werkzeuge oft zu mächtig für den Informatikunterricht erscheinen, dass ihr Einsatz zumindest auf den ersten Blick schwierig wirkt, und dass ihr Mehrwert nicht bekannt ist. Dieser Artikel zeigt, dass eine Vielzahl von Vorteilen der Einarbeitung in eine neue Software überwiegen.

2 Einführung von Versionskontroll-Systemen im Unterricht

Dieses Kapitel zeigt anhand von Beispielen für den Unterricht einen Weg auf, Versionskontroll-Software im Unterricht schrittweise einzuführen. Dabei werden einzelne Funktionalitäten bereits im (nicht projektorientierten) Unterricht eingesetzt, um einerseits auch dort Nutzungsvorteile zu haben und andererseits im Projekt schon auf Kompetenzen im Umgang mit Versionskontrolle zurückgreifen zu können. Damit wurden in der Jahrgangsstufe 11 eines bayerischen Gymnasiums sehr positive Erfahrungen gemacht.

Eine Durchführung bereits in Jahrgangsstufe 10 wird als unproblematisch gesehen. Zu beachten ist jedoch, dass in einem Abschnitt das Vererbungskonzept Voraussetzung ist.

Auf technische Aspekte wird erst in Kapitel 3 eingegangen. Vorausgeschickt wird hier nur, dass die Softwareentwicklung in der Sprache Java mit der Entwicklungsumgebung BlueJ durchgeführt wurde.

2.1 Der erste Kontakt mit einem Repository: Austeilen von Dateien

Im Informatikunterricht stellt die Lehrkraft Schülerinnen und Schülern oft Quelltexte zur Verfügung, z.B. in Form von Programmieraufgaben oder Musterlösungen. Eine Möglichkeit der Verteilung ist die Ablage in einem Ordner des Schulnetzes, auf den aber meist kein Zugriff von zuhause aus möglich ist. So müssen Lehrer und Schüler einen Transfer per USB-Stick oder Internet durchführen. Also macht man als Lehrkraft häufig die Dateien online zugänglich, etwa durch eine Lernplattform wie Moodle. Dem Vorteil des jederzeitigen Zugriffs steht der Nachteil eines höheren Aufwands beim Aktualisieren (etwa bei nachträglichen Verbesserungen) gegenüber, weil die Daten meist zum Hoch- bzw. Herunterladen in einen Datencontainer gepackt bzw. extrahiert werden müssen.

Hier bietet sich das Arbeiten mit Subversion an: Code kann unmittelbar aus BlueJ heraus aktualisiert oder heruntergeladen werden. Für die Lehrkraft ist der Aufwand minimal: Ist zu Hause erst einmal ein komfortabler Client installiert (s. Kapitel **3.4**), markiert die Lehrkraft einfach, welche Verzeichnisse in das Repository hochgeladen oder dort aktualisiert werden sollen und welche nicht (s. Abb. 2). Die Schülerinnen und Schüler benötigen lediglich die Adresse des Subversion-Servers; bei einem öffentlichen Lese-Zugang brauchten sie weder Benutzernamen noch Passwort noch Anmeldung. Diese sind erst dann erforderlich, wenn die Schüler auch eigenen Quelltext zum Projekt beisteuern.

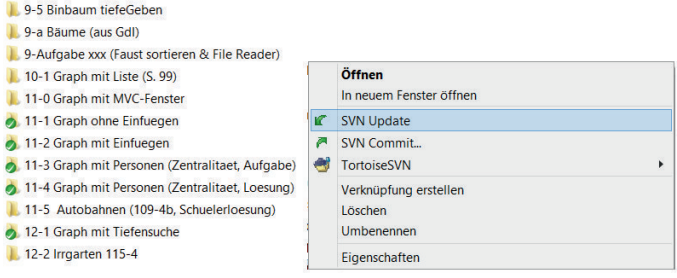

Abb. 2: Screenshot des Dateisystems des Lehrers – Repository Freigabe einzelner Ordner

2.2 Teamarbeit erleben – eine erste Zusammenarbeit mit einem Repository

Im zweiten Schritt bei der Einführung einer Versionskontroll-Software sollen die Schülerinnen und Schüler nicht nur eigenen Quelltext in ein Repository hochladen, sondern

auch erleben, wie man durch Zusammenarbeit im Team sehr schnell eine größere Aufgabe lösen kann – einer der zentralen Vorteile dieses Werkzeuges im Einsatz.

Die Vorgehensweise ist denkbar einfach. Der Lehrer bereitet ein (BlueJ-)Projekt vor, in dem eine Schnittstelle in Form eines Interface oder einer Oberklasse vorgegeben ist. Dazu erstellt jede Schülerin und jeder Schüler eine Implementierung bzw. eine Unterklasse. Das Interface „Tier“ verlangt beispielsweise von den implementierenden Klassen eine Methode „String lautGeben()“. Zu Beginn ist das mittels „Aktualisieren/Update“ aus dem Repository heruntergeladene Projekt noch relativ leer, es enthält nur das Interface und eine Testklasse (s. Abb. 3 links).

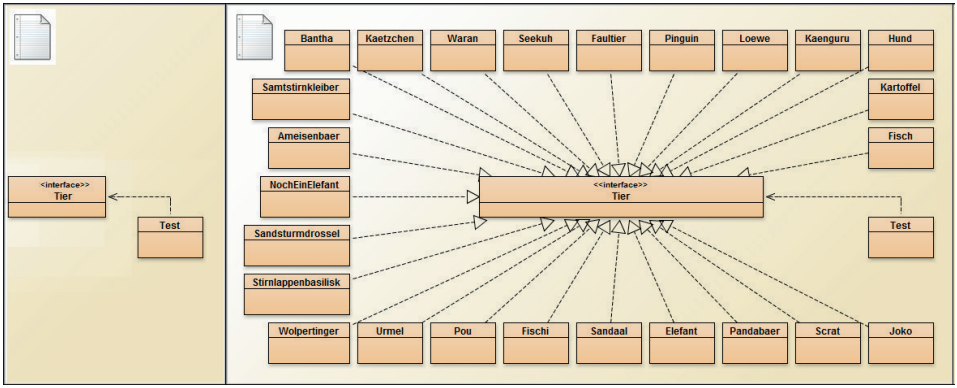

Abb. 3: BlueJ Projekt „Tier“ vor und nach der Teamarbeit

Nachdem jeder Schüler eine eigene Klasse erstellt und mittels „Abgeben/Commit“ an den Server übertragen hat, ist das Projekt mit einer großen Zahl von Klassen gefüllt – in kurzer Zeit hat man zusammen eine Vielzahl an Ausgestaltungen der Oberklasse geschaffen (s. Abb. 3 rechts). Dabei können die Schülerinnen, wenn sie regelmäßig Updates durchführen, verfolgen, welche Klassen bereits erstellt worden sind. Bei jedem Update können und sollen Schüler kommentieren, was sie am gemeinsamen Codeprojekt geändert bzw. ergänzt haben. Das ist bei größeren Projekten ein wichtiger Bestandteil der Dokumentation des Arbeitsprozesses.

Im Vergleich zum ersten Schritt – das Herunterladen und Aktualisieren von Projekten unter BlueJ/Subversion – kommt wenig hinzu, weil bei der Konzeption darauf geachtet wurde, dass es keine Kollisionen gibt: Da jeder Schüler seine eigene Klasse schreibt, kann es nicht zu Konflikten kommen. Anders als beim reinen Update brauchen die Schüler für einen Commit jeweils eine – möglichst eigene – Benutzerkennung für den Serverzugang (technischen Voraussetzungen s. Kapitel 3.3).

2.3 Konflikte beim Zusammenführen von Quelltextänderungen

Im dritten Schritt sollen die Schüler erfahren, welche Probleme es beim gleichzeitigen

Bearbeiten von Dateien geben kann. Dazu wurde vom Lehrer ein (BlueJ-)Projekt mit einer Klasse „Held“ angelegt, welche eine große Anzahl von Attributen enthielt. Im Paar arbeitend sollten die Schüler sich jeweils zunächst über eine Aktualisierung das Projekt vom Server herunterladen und danach noch nicht vorhandene Getter- und Setter-Methoden zu jedem Attribut anlegen. Wie vorhergesehen kam es dabei zu Konflikten. Schematisch fand dabei folgendes Vorgehen statt:

1. Schülerin A aktualisiert Projekt aus dem Repository.
2. Schüler B aktualisiert Projekt aus dem Repository.
3. Schülerin A legt Methode x an
4. Schülerin A führt Commit durch.
5. Schüler B legt parallel dazu eine gleichnamige Methode x an.
6. Schüler B versucht Commit durchzuführen.

In diesem Fall kann der Commit von Schüler B nicht durchgeführt werden. Er erhält stattdessen eine Warnmeldung; „Ihre Arbeitskopie ist nicht mehr aktuell. Holen Sie eine Aktualisierung aus dem Repository bevor Sie ihre lokalen Änderungen abgeben können.“

Schüler B führt also eine Update durch und erhält dabei, wie sonst auch, eine Nachricht über die zu aktualisierenden Dateien. Die in Konflikt stehende Datei ist mit dem Stichwort „Zusammenführen“ versehen (Abb. 4 links).

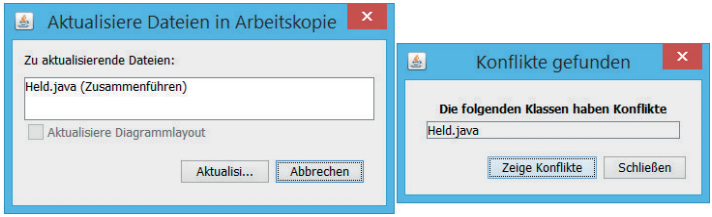

Abb. 4: Konfliktmeldungen

Setzt Schüler B die Aktualisierung fort, gibt es zwei Möglichkeiten: Vielleicht kann der Subversion-Client den Konflikt selbstständig auflösen. Das ist etwa dann der Fall, wenn Schülerin A und Schüler B an verschiedenen Methoden gearbeitet haben. Dann erhält Schüler B eine Code-Version, die seine eigenen Änderungen in den von Schülerin A geänderten und hochgeladenen Code integriert. (Den eigentlichen Commit dieser neuen Fassung muss Schüler B allerdings noch vornehmen.)

Oft kann der Konflikt aber nicht automatisch gelöst werden. In diesem Fall erhält Schüler B eine weitere Warnmeldung (Abb. 4 rechts), worauf er sich die Konflikte anzeigen lassen kann. Dabei wird ihm im Editorfenster von BlueJ der Code mit den in Konflikt stehenden Fassungen gezeigt: Oberhalb einer Reihe von ======= steht der Code des Schülers (bis hin zur Zeile „<<<<<<< .mine“), unterhalb steht der Code aus dem Repository (bis hin zur Zeile „>>>>>>> .r187“, wobei die Zahl die Revisionsnummer, zu der die Änderung gehört (Abb. 5). Jetzt kann Schüler B den Konflikt manuell lösen und die

verbesserte Fassung in das Repository hochladen.

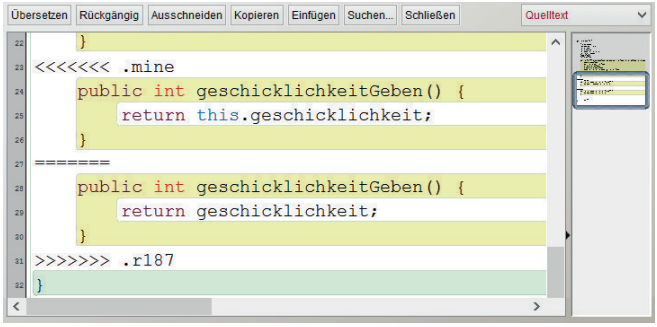

Abb. 5: Anzeige des Konfliktes im Quelltext

Es fällt (Schülerinnen und Schülern) nicht leicht, diese Konflikte zu lösen. Deshalb ist ein wesentliches Lernziel dieses Schritts, ein Bewusstsein dafür zu schaffen, dass bei einem kollaborativen Arbeiten an Quelltext Absprachen (z. B. über ein Taskboard) stattfinden sollten. Arbeiten die einzelnen Teammitglieder an unterschiedlichen Methoden in der Klasse, übernimmt wie oben geschildert die Versionskontroll-Software das Zusammenführen. Dies ist eine erhebliche Erleichterung im Projektablauf (vgl. Ausblick in [Br15]). In diesem Kontext ist auch eine Reflexion über Workflow wichtig (s. Abb. 6).

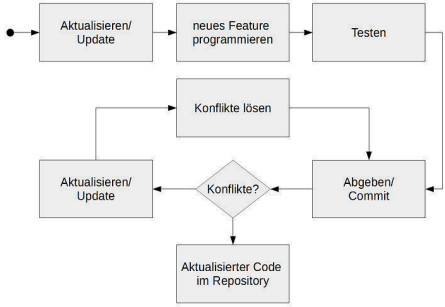

Abb. 6: Workflow für das Arbeiten mit einem Subversion Repository

2.4 Konfliktfreies Zusammenführen durch Teamabsprachen

Im nächsten Schritt sollten die Schülerinnen und Schüler an einem konkreten kleinen Programmierprojekt arbeiten, das Absprachen nötig macht. Ziel war ein schlichtes, funktionierendes Tic-Tac-Toe-Spiel nach dem bereits eingeführten Model-View-Controller-Muster. Dazu einigten sich die Schülerinnen und Schüler erst auf gemeinsame Schnittstellen in Form von abstrakten Model-, View- und Controller-Klassen (s. Abb. 7). Diese Schnittstellen wurden als BlueJ-Projekt allen Schülern über das Repository zur Verfügung gestellt. Zu diesen Klassen programmierten dann Schülerteams konkrete Unter-

klassen, und zwar so, dass es jeweils drei verschiedene konkrete Ausformungen zu allen drei gegebenen Schnittstellen-Klassen gab. Insgesamt arbeiteten also neun Programmierteams am selben BlueJ-Projekt. Wenn sich alle Teams an die Vorgaben der abstrakten Oberklassen hielten, sollten die Ergebnisse beliebig kombiniert werden können.

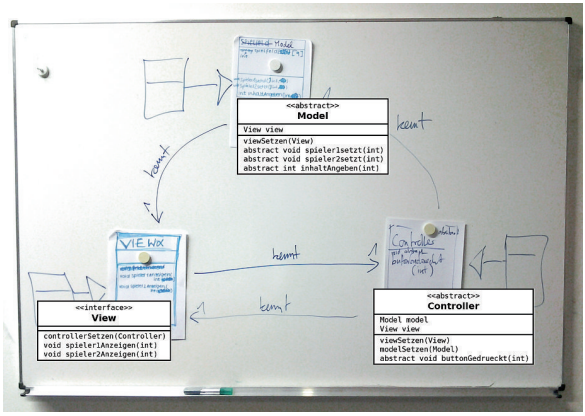

Abb. 7: Ergebnis der Schnittstellendiskussion zu MVC

Da es keine Java-Klasse gab, an der mehrere Gruppen gleichzeitig arbeiteten, konnten keine manuell zu lösenden Konflikte beim Abgeben der Dateien im Repository entstehen. Zwei Regeln stellten sich dabei als hilfreich heraus: a) jede Gruppe arbeitet wirklich nur an der eigenen Klasse; auch zur Lösung von vermeintlichen Fehlern darf nicht in anderen Klassen Code geändert werden, und b) nur fehlerfrei compilierender Code darf ins Repository geladen werden. Es stört die anderen Gruppen beim Arbeiten, wenn Fehlermeldung beim Übersetzungsvorgang erscheinen, und sei es auch in anderen Klassen.

Am Ende entstand ein BlueJ-Projekt mit drei verschiedenen Unterklassen von Model, View und Controller. Die Schülerinnen und Schüler erkannten dabei, dass sich die einzelnen Bestandteile des Spiels – solange sie sich an den vorgegebenen Schnittstellen orientieren – tatsächlich unabhängig von den anderen Teilen einsetzen lassen.

3 Technische Hintergründe und Details

3.1 Grundsätzliches zu Subversion und Git

Für das Verständnis von Subversions-Software sind Hintergrundinformationen zur Funktionsweise hilfreich, insbesondere weil konzeptionelle Unterschiede zwischen den Systemen Subversion [PC09] und git [Git15] [Ch09] existieren.

Bei Subversion sind auf einen zentralen Server in einem Repository (englisch: Depot, Magazin, Lager) alle aktuellen und früheren Dateiversionen gespeichert, ebenso alle

Änderungen und Zusatzinformationen. Die Benutzer arbeiten jeweils mit einem Subversion-Client: In einem Update-Vorgang laden sie diejenigen Dateien des Projekts vom Server herunter, die dort aktueller sind als die lokal gespeicherten. Danach werden die Dateien lokal bearbeitet, und in einem Commit-Vorgang werden die geänderten Dateien auf den Server hochgeladen. Ein anderes Teammitglied kann sich danach – oder auch gleichzeitig – ebenfalls mit „Update“ auf den aktuellen Stand bringen lassen, eigene Änderungen vornehmen, und diese mit „Commit“ an den Server weitergeben. Überschneidet sich das Arbeiten zweier Teammitglieder an derselben Datei, liegt ein Konflikt vor, den Subversion selbstständig zu lösen versucht bzw. anzeigt (vgl. Kapitel 2.3).

Auch bei Git wird mit Repositories gearbeitet. Allerdings speichert hier jeder Client eine vollständige Kopie des gesamten Repository, so dass letztlich alle Git-Repositories gleichberechtigt sind. Die Intention dahinter ist eine hohe Sicherheit gegen unbeabsichtigte bzw. böswillige Verfälschung. Die Hauptfunktion des zentralen Servers ist der Datenaustausch, nicht die Datensicherheit.

3.2 BlueJ und Subversion

Die in Schulen oft eingesetzte Java-Entwicklungsumgebung BlueJ enthält bereits einen funktionierenden Subversion-Client, so dass keine zusätzliche Software installiert werden muss – weder in der Schule, noch bei den Schülern zu Hause. Deshalb wird hier nur das Arbeiten mit Subversion (und BlueJ) vorgestellt.

Die Teamwork-Einstellungen von BlueJ müssen im Menüpunkt „Werkzeuge/Einstellungen“ erst freigeschaltet werden. Danach erscheinen am linken unteren Rand die Knöpfe „Aktualisieren...“, „Abgeben...“ und „Status“; außerdem enthält das Menü „Werkzeuge“ nun ein weiteres Untermenü „Teamarbeit“. Diese Knöpfe entsprechen den Aktionen „Update“ und „Commit“. Bevor man sie verwenden kann, muss sich BlueJ allerdings erst mit einem Repository verbinden. Dazu sind Konfigurationen unter dem Menüpunkt „Werkzeuge/Teamarbeit/Teamarbeitseinstellungen...“ nötig:

- Benutzernamen und Passwort – obwohl dies für ein öffentlich zugängliches Repository beim Update keine Rolle spielt, verlangt BlueJ eine Eingabe im Benutzerfeld
- Servertyp: Subversion – daneben unterstützt BlueJ noch das ältere CVS
- Server bzw. Verzeichnis – die Adresse des zentralen Subversion-Servers und der Pfad ins jeweilige Verzeichnis; ein Beispielprojekt ist unter der Adresse svn.code.sf.net/p/q11/code/trunk/ öffentlich zugänglich
- Protokoll: https

Gibt man diese Daten ein, wird man über den Menüpunkt „Werkzeuge/Teamarbeit/Arbeitskopie erstellen...“ mit dem Server verbunden und kann sich anzeigen lassen, welche Projekte dort zur Verfügung stehen. Das ausgewählte Projekt wird anschließend in ein Verzeichnis des lokalen Rechners übertragen. Wie man als Lehrkraft ein Repository einrichtet und komfortabel verwaltet, wird in Kapitel 3.4 erklärt.

3.3 Anlegen eines Repository

Zentrales Element bei Apache Subversion ist der Server. Subversion ist unter der freien Apache-Lizenz 2.0 veröffentlicht, so dass es sich leicht im Schulnetz installieren lässt (z.B. [svn15]). Da ein Zugriff von außerhalb der Schule gewährleistet werden sollte, wird man aber lieber auf gehostete Subversion-Installationen zurückgreifen.

Sehr komfortabel war das Project Hosting von Google, das unter anderem Subversion anbietet, und mit dem die Projekte in der 11. Jahrgangsstufen durchgeführt wurden. Zum einen hätte ohnehin jeder beteiligte Schüler bereits über einen Google-Account verfügt, zum anderen ist das Passwort, das für den Zugang zum Projekthosting verwendet wird, ein anderes als das zentrale Passwort des Google-Accounts. So war es problemlos möglich, Dummy-Accounts anzulegen, die von der Lehrkraft verwaltet wurden; die Schülerinnen und Schüler erhielten lediglich den Benutzernamen des Dummy-Accounts und das Google-Code-spezifische Passwort. Leider stellt Google sein Projekt-Hosting ab Mitte 2015 ein, so dass für spätere Projekte andere Lösungen gefunden werden müssen.

Bei kostenlosen Hosting-Anbietern wie SourceForge ist das Projekt mit allen seinen Dateien grundsätzlich öffentlich einsehbar; es steht stets unter einer Open-Source-Lizenz. Das hat den Vorteil, dass Schülerinnen den Projektcode auch ohne Zugangsdaten aus dem Repository laden können. Der Nachteil ist, dass Schüler und Lehrkräfte automatisch Nutzungsrechte an ihrem geistigen Eigentum einräumen. Außerdem darf kein Material im Rahmen des Projekts verwendet werden, für das keine Nutzungsrechte eingeräumt wurden, insbesondere urheberrechtlich geschütztes Material aus Schulbüchern.

3.4 Workflow für die Lehrkraft

Für Schülerinnen und Schüler reicht es, wenn sie mit dem in BlueJ integrierten Subversion-Client arbeiten. Als Lehrkraft möchte man aber mehr machen, als nur aktuelle Dateien vom Repository zu laden und geänderte Dateien dort zu aktualisieren. Zu diesem Zweck gibt eine große Zahl von Subversion-Clients mit graphischer Benutzeroberfläche. Für das Arbeiten unter Windows ist TortoiseSVN besonders geeignet. [To15] Nach der Installation erweitert TortoiseSVN das Kontextmenü standardmäßig um drei Einträge (s. Abb. 2). Für ein Windows-Verzeichnis, das mit einem bestehenden Repository-Projekt verbunden ist, können damit Updates und Commits durchgeführt werden.

Das erste Herunterladen eines bereits existierenden Repository mit TortoiseSVN geht in zwei Schritten: Erst legt man einen Ordner an, und wählt nach einem Klick mit der rechten Maustaste aus dem Kontextmenü „SVN Checkout...“ aus. Im nächsten Schritt gibt man die URL des Repository an bestätigt die Auswahl. Danach werden die aktuellsten Fassungen aller Dateien aus dem Repository heruntergeladen. In Zukunft arbeitet man auf diesem Ordner dann nur noch mit den Menüeinträgen zu Update und Commit – bei letzterem muss man natürlich die Zugangsdaten zum Repository angeben.

In die eben heruntergeladene Kopie des Repository kopiert man die BlueJ-Projekte, die

man für einen Informatikkurs braucht. Manche davon möchte man vielleicht nicht gleich mit dem ersten Commit den Schülern und Schülerinnen zur Verfügung stellen. Deshalb kann man mit der rechten Maustaste im TortoiseSVN-Kontextmenü auswählen, welche Ordner mit dem Repository synchronisiert werden sollen und welche nicht. Nur die Inhalte der mit einem grünen Häkchen versehenen Ordner werden im Repository gehalten, die anderen Ordner markiert man mit „Unversion and add to ignore list“ (s. Abb. 2). Das kann auch nachträglich jederzeit geändert werden.

4 Fazit und Ausblick

In diesem Artikel wurde aufgezeigt, dass der Einsatz von Versionskontroll-Systemen im Informatikunterricht sowohl bei der Einführung in die Programmierung, als auch in Softwareprojekten einen Mehrwert hat, weil Workflows im Unterricht unterstützt werden, durch einen verteilten Zugriff und automatisches Zusammenführen von Quelltest kollaboratives Arbeiten im Team deutlich erleichtert wird, sowie die Dokumentation des Arbeitsfortschritts möglich ist.

Bei der technischen Umsetzung fand mit der Auswahl der Teamwerkzeuge von BlueJ eine starke Einschränkung statt. Vorteile dieser Umgebung sind, dass ohne ein zusätzliches Programm der Zugriff auf das Repository direkt aus der Entwicklungsumgebung möglich und die Bedienungsmöglichkeit auf das Notwendigste reduziert ist. Dadurch ist die Einarbeitungszeit und Komplexität für den Bediener minimal. Dem steht der Nachteil gegenüber, dass nicht alle Funktionalitäten eines Repository, z. B. das Arbeiten mit Branches, verfügbar sind und auch die Sicht auf das Repository eingeschränkt sind.

Möchte man hier einen anderen Weg gehen, bietet sich als svn-Client Tortoise bzw. als git-Client SourceTree an. Letzteres visualisiert grafisch auch unterschiedliche Entwicklungen und Zusammenführungen. Dazu haben die Autoren noch keine Unterrichtserfahrung. Leistungsstarke Oberstufenschüler hatten jedoch bei einem Praktikum in einem Softwareunternehmen keine Probleme damit.

5 Literaturverzeichnis

[Br15] Brichzin, P.: Agile Softwareentwicklung, Tagungsband INFOS 2015, Darmstadt.

[Git15] Offizielle git Homepage, http://git-scm.com/ (zuletzt geprüft am 27.04.15).

[Ch09] Chacon, S.: Pro Git, Apress; 2009 http://git-scm.com/book/de/v1 (zuletzt geprüft am 27.04.15).

[PC09] Pilato, M.; Collins-Sussman, B.: Versionskontrolle mit Subversion, O’Reilly, 2009.

[Svn12] Subversion for Windows with Apache server (zuletzt geprüft am 27.04.15).

[To15] Offizielle TortoiseSVN Homepage, http://tortoisesvn.net (zuletzt geprüft am 27.04.15).

Agile Projects in High School Computing Education – Emphasizing a Learners' Perspective

Ralf Romeike
University of Potsdam
August-Bebel-Str. 89
14482 Potsdam, Germany
romeike@cs.uni-potsdam.de

Timo Göttel
University of Hamburg
Vogt-Kölln-Str. 30
22527 Hamburg, Germany
tgoettel@acm.org

ABSTRACT
Software projects are seen as a methodology for secondary computing education which is highly appropriate and meets the demands and goals of Computer Science (CS). Yet the majority of models and examples for project-based lessons rely on a traditional software development approach: the waterfall model. In this paper such models are analyzed for their strength, problems, and deficiencies. Based on the results of the analysis a new approach to projects in secondary computing education is presented which uses the concept of didactic transposition to adapt agile software development methods for project organization, management, and implementation in class. The resulting model applies valuable practices of Extreme Programming and Scrum and provides a set of tools that allow high school software projects to benefit from modern software development methods. By emphasizing dynamic processes and a clear course of action an attractive perspective on CS is promoted.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *computer science education*.

General Terms
Human Factors, Theory.

Keywords
Secondary computing education, Agile methods, Project-based learning.

1. INTRODUCTION
In secondary computing education, software projects are promoted to provide an appropriate and student-oriented approach to Computer Science (CS) [19; 23; 32; 39]. Yet, most projects in this context are mainly focused on sequential project layouts that resemble traditional software development (SD) methodologies such as the waterfall model. In recent years it became apparent in professional SE that such methodologies often fail to produce high quality products, bring forward delays in delivery, and insufficiently consider customers' needs (e.g. [26]). Analogue issues can be found in school projects: unfinished projects, missing time and motivation for testing, neglected documentation, and teachers' difficulties in managing software projects are just some of the problems reported (cp. [19; 24; 32]). In modern SD, agile methods are promoted to provide a dynamic project management that relies on interaction and short design iterations. Agile methods build upon values and provide practices that are also highly expedient in high school contexts. Therefore, we present a new approach to projects in secondary computing education, which implements the theory of didactic transposition to adapt agile methods for project organization, management and implementation in classroom. Valuable agile practices of Extreme Programming (XP) [2] and Scrum [39] will provide a set of tools allowing software projects in high schools to reference modern SD by highlighting dynamic processes that help to focus on good results, a clear course of action, and an attractive perspective on professional CS by addressing common problems at the same time. In section 2, research on projects in computing education will be discussed and problems with prevalent models will be analyzed. The findings suggest a missing consideration of the learners' perspective in project models. In section 3, agile methods in professional and educational settings are discussed for their potential of supporting a learners' perspective. By describing the agile model for school projects in computing education common agile practices are characterized and adapted. Finally, the model is discussed in the context of existing models, its potential, and issues in computing education.

WiPSCE '12, November 8-9, 2012, Hamburg, Germany.

2. PROJECTS IN EDUCATION

2.1 Project Based Learning
Project-based learning (PBL) is an approach to teaching and learning in the classroom aiming for engaging students in explorative and problem-solving activities in authentic contexts. The concept is described to originate from teaching in Zurich and Paris in the 19th century. Since then, the idea has been picked up by various teachers and researchers and enriched with psychological, pedagogical and sociopolitical aspects, e.g. by Dewey und Kilpatrick [11]. Projects are understood as learning processes that draw on interests and demands of the students by striving for a complex result, often a product. This includes planning, problem-solving, analysis of different solutions and the evaluation of the process and its product. PBL is known for increasing students'

motivation, for strengthening self confidence, and for fostering satisfaction in process and outcome (cp. [5]). Therefore, it is promoted to foster high-level thinking skills including problem-solving and analysis skills. Thus, it helps to gain a deep understanding of topics and processes (cp. [1; 25]). PBL is known to encourage peer interaction (cp. [25]).

In Germany, Frey [16] elaborated PBL by describing the process along the following steps: A project starts with a project idea, which should be based on the interests of the students. Subsequently the idea is specified in order to find agreement on what the class will be trying to achieve. After planning the necessary activities the project plan is carried out. Milestones and meta-communication serve as tools for supporting the process. This model is commonly accepted within Germany and was fundamental for an adapted model in German computing education (cp. 2.3)

Summarizing, projects are characterized by being problem-focused and interdisciplinary, by allowing students a choice of topic and foster personal responsibility. Projects are generally carried out over a longer time span and are often graded "differently", e.g. by focusing more on processes and applying less pressure.

2.2 PBL in Secondary Computing Education

Because of the above mentioned benefits, PBL is widely used in higher computing education (cp. [6; 13]) and secondary computing education, especially in the context of software projects [39; 41]. Consequently, PBL is recommended as an appropriate approach to computing education in the majority of German curricula. However, there is limited research focusing on methodologies for SD projects in secondary education. Meerbaum-Salant and Hazzan [33] constitute "as far as we know, no general methodology has been developed for software development projects in the high school". Consequently, they propose an approach which focuses on a mentoring model for teachers (cp. 2.4).

In Germany, a model for high school software projects was proposed by Frey [15] and elaborated by Schubert and Schwill [39]. The majority of published school SD projects can be attributed to this model. Therefore the model will be analyzed in the following.

2.3 A Professional Perspective

A majority of publications concerning the use of projects in secondary computing education[1] stems from the 1980s and 1990s, generally analyzing and adopting the professional approach to SD and adopting the waterfall model for the classroom. The idea was that a professional model for SD would provide an appropriate framework for school projects: it offers students a structured learning process and gives insights into professional SD processes at the same time. In comparison to other subjects using PBL only as a teaching method without a connection to the subject matter itself, projects are scientifically anchored in CS [39]. Consequently, there is a large body of practice reports on SD projects. However, by analyzing publications of the major German conferences in computing education of the last 10 years we could not find any publication concerning methodologies for school projects in general secondary education. However, research shows that teachers consider it as important that students get familiar with SD processes, e.g. by running "through the workflows of the waterfall model" [30].

Figure 1 illustrates the project steps of Schubert and Schwill's model [39] with the corresponding output of each phase. The proposed model describes student activities along the software life cycle. However, it does not provide methods or practices of how students can reach the expected outcome of each phase. In the problem analysis phase all important environmental conditions need to be gathered clearly and completely. Furthermore, this phase includes planning activities for time, team and equipment. The resulting requirements definition serves as a contract between teacher and students. In the subsequent system design process a model of the system is specified by dividing the "overall system" into modules. Until here, the activities shall be performed by the full team, which is allowed to split up into subgroups for minor tasks. Then, smaller groups handle a module each under their own responsibility. The remaining phases follow the software life cycle.

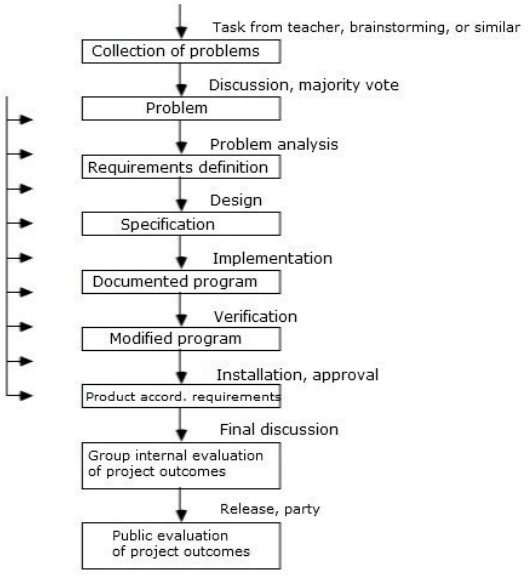

Fig. 1. Project model by Schubert and Schwill [39].

Concerning the team structure, Schubert and Schwill recognize that the organization of the team cannot follow hierarchical structures as typically used in professional SD teams. Instead, they suggest equal status and responsibility among team members and a "force" for communication and common goals. This goal is addressed by assigning eight different roles for student positions within the project (computer responsible, project supervisor, interface responsible, tester, documentation responsible, butler, session chair, secretary).

Criticism of the methodology points out problems with a perceived bureaucratic overhead: "It is always the same: Students refuse to first plan on paper. Because only small programs are written, these are not documented. The taught principles of software development are hardly noticed" [19]. Often, testing is omitted in the project realization due to a lack of time and a lack of perceived importance, especially if the software does not have a practical use after the project (cp. [24]). Other problems may result from the structure of the project: The time span that needs to be scheduled is up to half a year (Schubert and Schwill [39] suggest to perform one project per semester). This is difficult to plan

[1] Research on this topic is rare. The statement reflects the situation in Germany. However, we are not aware of comparable publications from other countries.

for, especially if students lack project experience and supportive practices. Additionally, a sequential project layout collides with "project-unfriendly" circumstances of formal lessons such as limited time, heterogeneous student abilities and lessons spread over several weeks. Humbert [23] even summarizes that the pedagogical dimensions of PBL are not sufficiently considered in such a project model. Additional problems of conducting school SD projects are outlined in the following by pursuing a teachers' perspective.

2.4 A Teachers' Perspective

Meerbaum-Salant and Hazzan [32] analyzed difficulties encountered by teachers in mentoring SD projects in Israeli high schools. Even though the curricular background and objectives are somewhat different[2] than for projects in general educational settings as described in this paper, the results are similar to the problems reported from German teachers. Additional problems were identified in the contexts of scheduling the project, CS expertise of the teachers, considering students' individual performances, and evaluation of the project. Teachers describe mentoring of SD projects as a more complex task compared to traditional teaching.

Meerbaum-Salant and Hazzan [33] express the need for a general methodology for SD projects in high school and address the previously identified problems in a mentoring methodology (ACMM). It is intended to support teachers, who are expected to be confident in a variety of knowledge types [32]. Therefore it describes a set of practices (Pedagogical Class Management Aspect, Social Aspect, Project Management Aspect) that shall be considered by teachers while mentoring a project. The ACMM takes into account the principles of agile software (such as communication, simplicity, feedback, respect), which basically are reflected in the teacher-student interaction.

2.5 A Learners' Perspective

For learning settings, where a team of students is working cooperatively on projects, we see potential for taking the idea of applying agile methods further than it is described in the ACMM: project management can be done by the student team. This can be supported by straightforward and easy to use methods adopted from modern SD. Additionally, students may benefit from experiencing a SD process which also includes management aspects in addition to activities like analysis, designing, coding, and testing, as described in the other models. In the following we demonstrate how *problems* identified by Meerbaum-Salant and Hazzan [32] may be addressed in such a project by considering agile methods as presented in section 3.

Schedule: Teachers may need to catch up with teaching of material during the project. The sequential project approach does not allow for such a teacher's intervention without disturbing the process. However, in an iterative project design, issues and success can be discussed in class regularly.

Required CS knowledge: Some teachers admit a lack of project development knowledge. Students will need help with CS knowledge while solving problems. It meets the ideas of PBL if student teams would be empowered to manage projects themselves. This can be supported by easy to follow practices and strategies which make teacher involvement almost unnecessary. Additionally, clearly defined practices may support teachers' confidence. Heterogeneous student teams stimulate mutual assistance before requiring the help of a teacher.

Students' individual work: Teachers see a need for personal supervision in order to achieve a timely completion of the project and meeting of the requirements. Agile methods allow for transferring this responsibility to the project team, hence relieving the teacher.

Evaluation of project outcomes: Agile practices naturally lead to a variety of documents which can be considered for project evaluation (e.g. estimates in planning poker or burn-down charts). Furthermore mutual assessment within teams may be performed.

All suggested practices require a change of the project perspective from the teacher to the learner. In section 3 the mentioned agile practices are discussed in more detail and transferred to classroom settings by the use of didactic transposition.

2.6 Didactic Transposition for Project Methodologies

Didactic transposition describes the process of adapting professional knowledge of a domain for teaching scenarios based on a didactic intent [9]. Hazzan et. al. [21] applied didactic transposition on agile SD methods with the intent to create a teaching framework and a mentoring methodology for software projects.

The model for school SD projects discussed in 2.1 can be attributed to didactic transposition as well. Here, the professional process was adapted under consideration of the underlying principles of PBL. Schubert and Schwill [39] emphasize the advantage of a method which is learning activity and learning content at the same time. However, we see potential for shifting the focus from professional process *knowledge* to modern professional *methods* which may be adapted in a way that they address previously outlined problems in school SD projects.

In the following, we discuss agile methods in professional SD and in education. We applied didactical transposition for developing an agile approach to projects in computing education which emphasizes a learners' perspective.

3. AGILE PROJECTS IN COMPUTING EDUCATION

3.1 Agile Methods in Professional and Educational Settings

Agile methods are popular amongst researchers and practitioners for enabling software developers to create systems that are more likely to be accurate in meeting customers requests, finishing in time, building robust systems, and creating usable/readable code (cp. [22]). Therefore, in industry agile methods are currently replacing waterfall or other linear methodologies that are known for shortcomings in the above mentioned goals of SD. Agile methods are focused on social interactions and dynamic creative processes. Hence, developers in agile teams often report on a strong satisfaction in their work experience and strong confidence in their outcomes (cp. [27; 31]).

[2] German curricula emphasize computer science concepts in the context of general education which only partly includes algorithmic thinking and programming. In comparison, the underlying curriculum of the study emphasizes foundations of algorithmic thinking and programming [17]. Additionally, we understand projects as teamwork where several students or the whole class are working on the same goal, Meerbaum-Salant and Hazzan [31] describe projects where students work individually and "each student has his or her own project subject".

The agile manifesto of 2001 [3] clearly presents values contrasting traditional linear methodologies and underlying understandings:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

Agile methods are implemented in various frameworks. XP [2] and Scrum [40] are the most prominent implementations applied in industry and academia. Those methodologies define practices to assure compliance with the agile values as described in the agile manifesto. Agile methods are mostly understood to support a development process comprehensively from start to the final stage. However, the individual practices can be classified according to their main targets. Some practices are designed to structure team processes and customer collaboration while other practices focus on the quality of code and outcome.

Recently, several authors report on using agile methods in CS education at university level. Braught et al. [7] promote the use of agile methods because it helps female students to engage in programming tasks through interaction with peers. Nagappan et al. [34] highlight social experiences, learning processes, and quality of code when using agile methods in CS1 courses. However, literature shows that there may be possible barriers hindering an implementation of agile methods in university scenarios. Rico and Sayani [37] present a study where they found that students had already established their own approaches and habits for SD that were opposing practices of agile methods. According to Rico and Sayani it was almost impossible to convince the students to adhere to the introduced practices of agile methods. In this connection, they recommend to introduce agile methods as early as possible. Consequently, a benefit is assumed in the use of agile methods in school contexts. Literature on agile methods at university levels is still discussing the possibility of conducting a fully-fledged project management according to an implementation as XP or Scrum. Some authors promote an almost complete implementation of XP (e.g. [28]), some recommend to use and adopt a subset of practices (e.g. [8]), while others exclusively use one practice (e.g. pair programming) to support computing education (e.g. [7]). Schneider and Johnson [38] reviewed agile methods in computing education and highlight the importance of applying suitable practices according to their goals instead of fulfilling complete implementation of agile methods. Accordingly, Hazzan and Dubinsky [20] present ten reasons to consider agile methods in computing education.

Yet, literature on agile methods in secondary computing education is rare. Weigend [42] introduced elements of XP (user stories, spikes, test driven development, refactoring, and big visible charts) to provide a project-based iterative infrastructure that supports writing of high quality code. However, a sound methodology for connecting the presented elements in projects was not provided.

The work in hand is based on encouraging classroom experiences presented by Göttel [18]. In various educational CS projects, agile methods, such as pair programming, standup meetings, informative workspaces, and user stories supported students in their project work and additionally helped students to discover social aspects of CS. This success provided our basis for developing a comprehensive agile model for school projects in computing education.

3.2 An Agile Model for Projects in Computing Education (AMoPCE)

As discussed above, PBL represents a common teaching and learning method in computing education. However, even if common models suggest a structure and requirements for school software projects, methods are described insufficiently for the individual phases. Agile methods and modern SD principles provide a set of clearly described strategies that seems well suited for an implementation in school contexts. As described in the agile manifesto, they emphasize communication, visualization, teamwork and common goals. In the following, we want to introduce a model for school software projects that builds on the character of agile methods in order to address the problems outlined in section 2. It follows the agile manifesto by focusing on

1. Students and their interactions
2. Rapid success and working software
3. Collaboration in order to strive for a common goal over fulfilling a contract
4. Responding to change and learning progress over following a plan

The individual strategies and tools are illustrated in fig. 7 and will be described below in an agile model for projects in computing education (AMoPCE).

In this description we focus on processes and methods that are central for the agile SD process. Additional pedagogical aspects such as triggering students' motivation or finding agreement in choosing a project topic are not covered.

The process contains various techniques adapted from professional SD practices. They provide clear lines of action that can be followed by the students (e.g. generating user stories, planning poker, defining tasks). However, before applying them in a project we suggest introducing and practicing each method. On the other hand, an explorative learning approach is possible: The methods describe the processes in such detail that appropriate material can be created which allows students to learn and perform the processes independently.

The methods will be first discussed from a professional perspective[3] and subsequently transferred in a way that they can be *applied in classroom* (italic text). Examples will illustrate the process.

3.2.1 Preparation

Creating software requires competencies in programming and using tools. It is the responsibility of the teacher to make sure that the students have acquired the competencies needed for the software project ahead or that they will be able to acquire them during the project (e.g. with provided teaching material). Another aspect considers establishing an appropriate infrastructure (see [33] for further elaboration).

3.2.2 Ideas in

The initial steps of a SD process usually are devoted to requirements analysis listing possible features, approaches, and needs of

[3] In order to maintain a consistent presentation, methods and practices described in this paper are based on [35], which may be referred to for elaboration and additional information on modern SD practices.

the target audience. This phase is based on interviews, observations, and brainstorming sessions with the customers.

Building upon ideas and interests of students is a central characteristic of PBL. However, experience shows that students may not easily come up with ideas that can be implemented in such a project, especially at the first time. Therefore, an initial presentation of possible projects and the use of creativity techniques (e.g. brainstorming) are suggested. Resulting ideas shall be written down on individual Post-Its or cards for each activity that the software needs to provide (cp. fig. 2).

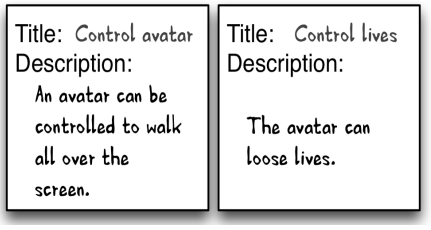

Fig 2. Post-its for recording ideas.

3.2.3 User Stories

User stories briefly describe features of a product that should be available to the actual user. Each user story addresses a specific activity of a user and is derived from the ideas of the requirements analysis. They are written from the perspective of a customer. User stories should easily fit on index cards and also be understood by non-developers. They should provide additional space for an estimation of the work effort.

In combination, user stories specify the entire intended product. A final state and amount of user stories has to be accomplished in agreement with the customer. Thereafter, stories are prioritized together with the customer by sorting user stories according to the importance of each story. Priorities are presented using incrementing numbers by powers of ten from 10 (most important) to 50 (least important).

Furthermore, the customer is asked to pick those stories that should be available in the initially delivered outcome or rather first major release. Consequently, discussion and reprioritizing stories may be necessary considering the basic features wanted for the first release.

User stories are created using various brainstorming techniques and take account of domain specific needs, knowledge, and approaches of the actual users specified in the requirements analysis.

A user story

- covers one activity that needs to be addressed
- represents the perspective of the customer
- is short, i.e. contains no more than three sentences
- does not use technical terms
- does not specify technology or tools

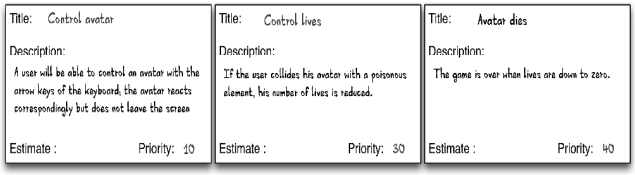

Fig. 3. Cards for user stories holding a title, a description, an estimate for workload, and a priority. Estimates will be added after completing the planning poker.

In professional SD projects one of the most important (and often unsuccessful) tasks is to find out what the customer wants. User stories provide a helpful way for achieving this goal. Since the students are going to implement their own ideas in their software projects, this goal does not apply. However, the team needs to find an agreement on the requirements for the software. These will be represented from a user's perspective: User stories briefly describe how a user interacts with the software. They can be developed from the previously recorded ideas. These need to be analyzed to find out, which interaction is really going to happen. This will be achieved by role playing and observation. Role playing is suggested as an attractive method for computing education to understand processes (cp. [4; 12; 14]). However, some of these examples use role plays in awkward contexts. In contrast, by role playing user interaction with the desired software system, students use a method which is anchored in modern SD processes and helps to identify relevant processes. The rules of the role play are simple: One student pretends to be the software and reacts accordingly. A sheet of paper may be used to illustrate the display. Another student takes the role of the user and instructs the software about what he or she wants to do, according to the previously obtained ideas. The remaining students observe the situation carefully to understand details and constraints of the desired product. The role play should be repeated several times with changing actors until no more new requirements arise. With this experience it should be easy to formulate the requirements from a user's perspective and write down the corresponding user stories (cp. fig. 3). Finally, the user stories receive a value for their priority. Priorities can be determined as a team, since generally agreement is quickly found.

Communication is an essential element of agile SD. Even if a set of user stories will now describe the final goal, questions and changes will appear in the following process. Since there is no customer who can answer questions and make decisions in order to clarify yet open questions, a group member needs to take over this special role: the product owner. This position may be passed around with each iteration.

3.2.4 Planning Poker

Planning poker is a hands-on method helping participants to estimate time needed for the work packages and guarantees a fair and comprehensible approach amongst all team members. Each participant holds a deck of cards to estimate the workload of a user story. There should be cards representing estimates in comprehensible units (e.g. developer-days) and special cards allowing players to indicate a lack of information, a need for a break, and already finished functionalities as shown in fig. 4.

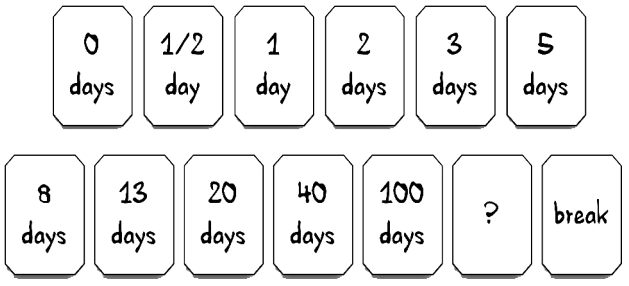

Fig. 4. Deck of cards used for the planning poker.

Each play round is devoted to one user story. A user story is placed in the middle of the table by the dealer and all participants place a card specifying their estimate face down on the table. All played cards are turned at the same time. The dealer collects the played cards and sums up the estimates trying to set up an average estimate. The dealer should address outliers by asking for reasons explaining fundamental differences in the estimates. Furthermore, the dealer should reflect on average estimates referencing the differences in the played cards. After each round the acquired estimate is written on the card of the user story. Additionally, the individual estimates are written on the back of the user story card to keep track of the decision process.

For a student team it is one of the most difficult tasks to estimate the workload and time demands of a given project due to a lack of project experience. Additionally, very likely not all planning relevant aspects are known at this point, learning processes will happen and changes may be necessary. Planning poker describes a playful way for challenging all students of the team to engage in the planning process by analyzing user stories and tasks, relating, estimating, explaining and defending their calculations, thus practicing their communication skills and ability to give and receive criticism.

For school software projects the same card values can be used as in professional SD. However, since these projects comprise a shorter working time, instead of days, 15 minute-periods seem to be appropriate. Each student estimates the time he or she believes he or she would need to implement the user story in focus. User stories should be presented with decreasing priority. Discussion of very divergent estimates will help resolving unspecified requirements and assumptions. After the planning poker is finished, the total workload for all user stories is divided by the number of programmers or programming teams (if pair programming is used) and compared with the time available. Unlike in professional SD, the priority is not to fulfill all requirements of the software but it is indispensable to keep the time available. If there is a major difference, e.g. more than 20 per cent, amount or complexity of the user stories need to be modified.

Again, for planning poker one team member needs to be the dealer. We suggest for this special role the position of a "teammaster", who leads the planning poker as well as team meetings. In the subsequent process this position also may be passed around with each iteration.

3.2.5 Tasks

After the initial planning poker, user stories are broken down into tasks. Usually each user story can be seen as a collection of tasks. A task is a rough description of a work package that should be done by a single developer. A task should have a unique self-explanatory name and should indicate its priority. The workload of each task should also be estimated by planning poker. Afterwards, task estimates are summed up to double check them with the initial estimates on the corresponding user stories. Thus, these estimates should not differ tremendously.

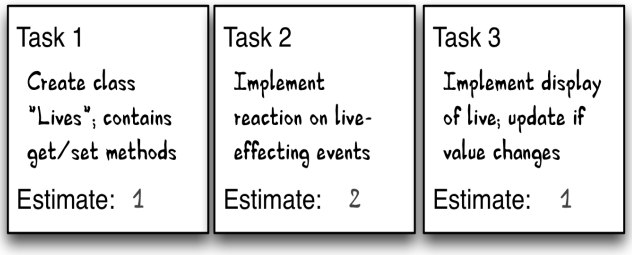

Fig. 5. Tasks for User Story "Control Lives".

While the user stories describe project goals from the perspective of the user, the students now need to change their perspective and look at them from a developer's viewpoint. By dividing user stories into tasks, various design decisions need to be made. Experienced programmers will now benefit from their competencies and known best practices, less confident students at this point can benefit and learn from team members, processes and team discussion.

3.2.6 Iterative Development, Prototypes, and Milestones

Agile processes are designed to provide short iterations that constantly come up with working prototypes that can be used and discussed with users or customers. This allows for rapid feedback loops that help to uncover misunderstandings, to detect issues in using the interface, and to adapt to new requests. An iteration is supposed to be short and has to be balanced according to implementing new features, fixing bugs, responding to change, and considering group dynamics or individual demands. In professional contexts, iterations vary between one week (5 working days) to one month (approximately 20 working days).

Iterations are planned in a team meeting by considering user stories' priorities, estimates, and the intended duration of an iteration. After deciding on the user stories to work on for the next iteration, they are pinned on a project board including associated tasks.

In school software projects, the planning of long development processes is reported to be difficult. Also, teachers report issues maintaining student motivation while they are not getting a grasp of the product until the whole project is assembled. The learning theory of constructionism emphasizes that learning happens especially felicitously in a context where learners are engaged with creating and investigating a personal relevant product (cp. [35]). Iterative development allows for creating a series of prototypes that can be analyzed, examined and played with in a constructionist sense. Also, such a design of the development processes allows for a higher flexibility in team organization and diversification due to more frequently changing tasks. Hence, each iteration is a mini-project containing each phase of the SD processes (requirements, design, code, test), but is easier to handle. This gives students the opportunity to perform the whole process several times within one project, to learn from and reflect on previous experiences and to take over several tasks in the team (in comparison to

other project models, where tasks are more strictly divided amongst students). Besides the iterations, which should not be longer than one to two weeks (which equals 2-4 lessons), milestones are used to structure the process and point out major achievements within the project. We suggest identifying 2-3 milestones for each project, representing versions of the final product with increasing value. However, only the achievement for the next milestone in the development is determined at a time. Milestones can be used for presenting the project progress for the rest of the class or teachers. Also, milestones should be positioned at times when the project pauses and teacher input is planned. Goals for the next milestone and the project progress are visualized at the project board.

3.2.7 Project Board

Project boards visualize goals and status of a current iteration and support target-oriented discussions. They present user stories and tasks in different status areas. Project boards are updated and discussed throughout the entire process. Thereby, it helps team members to keep track of the progress of the design process: the different areas of the board are used to present goals and accomplishments to the whole team. There are three main status areas: to-do user stories with associated tasks for the current iteration, tasks that are in progress, and completed tasks. Furthermore, there is an area to store user stories that need to be reconsidered in a future iteration. To provide a clear view, another area is reserved for finished user stories, allowing to take off corresponding task cards. Figure 6 presents an accordant project board.

Additionally, a burn-down chart is available on the project board. The chart visualizes the working time left in an iteration and work that needs to be done according to the task estimates. The chart is constantly keeping track of the progress by plotting the remaining sum of tasks at the end of a working unit.

Likewise, in a school software project all user stories with corresponding tasks are collected and presented at the team's project board. The project board is the central organizational and informative workspace for the entire project and should be available at all times, e.g. by placement at the classroom wall. It is also the meeting point for the regular standup meetings.

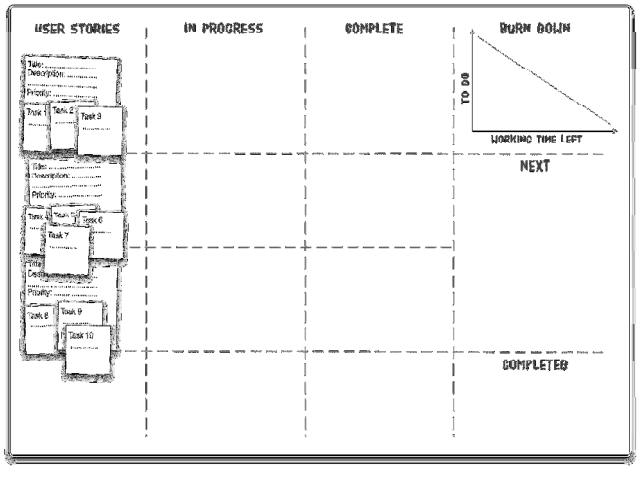

Fig. 6. Project Board including a burn-down chart.

3.2.8 Standup Meetings

Standup meetings provide a recurring fast and short update of the efforts of the team: Each team member has to report on accomplished tasks, possible issues in accomplishing certain goals, and a plan for the work day. Meetings are done while standing to guarantee a fast and goal oriented session kicking off a workday and should not exceed 5 to 15 minutes.

In school projects, standup meetings can provide an elegant way for starting off a lesson or working day within a project by encouraging team communication, sustaining motivation and identifying problems. The team gathers around the project board and recalls the project status, success and problems of the last working session and the goals for the day. After each team member has given a short statement the burn down chart is updated. If non-minor problems are identified, a longer meeting may be scheduled.

3.2.9 Pair Programming

Pair programming ensures an elaborated coding style: A pair of programmers uses a single programming environment for coding. The person using the keyboard and mouse is adopting the role of the "driver". The driver is actually coding and asked to present his or her ideas to the second programmer (the "navigator") verbally. Meanwhile, the navigator questions the coding outcomes, discusses possible misinterpretations, and seeks for alternative solutions that are more straightforward by keeping in focus the overall goals. The roles of driver and navigator are changed repeatedly during a workday. Programming in pairs helps to detect possible slips in the design and architecture of the code at early stages. Furthermore, it helps programmers to build upon social interaction uncovering misinterpretations of relations and intentions of code parts.

In many schools, two students share one computer due to limited hardware availability and hence often program in pairs. The agile method of pair programming supports this practice and adds a framework that encourages attention from both students, mutual learning and a notion of programming as a social activity.

Fig. 7 illustrates the organization of a school software project based on AMoPCE as described above. Other agile methods and ideas may be included in such a school project as well, e.g. test driven development, refactoring or "keep it simple".

3.3 Focusing on Programming Style and Outcome

Several agile practices may be applied in the process to bring forward a high quality outcome. However, since these practices are optional for project organization and partly depend on programming environments, in the following they are only summarized but not adapted for school software projects.

3.3.1 Test Driven Development

Test driven development replaces documentation and provides criteria to evaluate the code solutions: Programmers define intended functionalities by writing automatic tests covering all states and the correctness of the accordant results. First reports of using this method in secondary education have been published (e.g. [10]).

3.3.2 Refactoring

Refactoring introduces the idea that every part of the code should be reconsidered and changed if a more accurate solution can be found: Developers should rewrite parts of code without adding

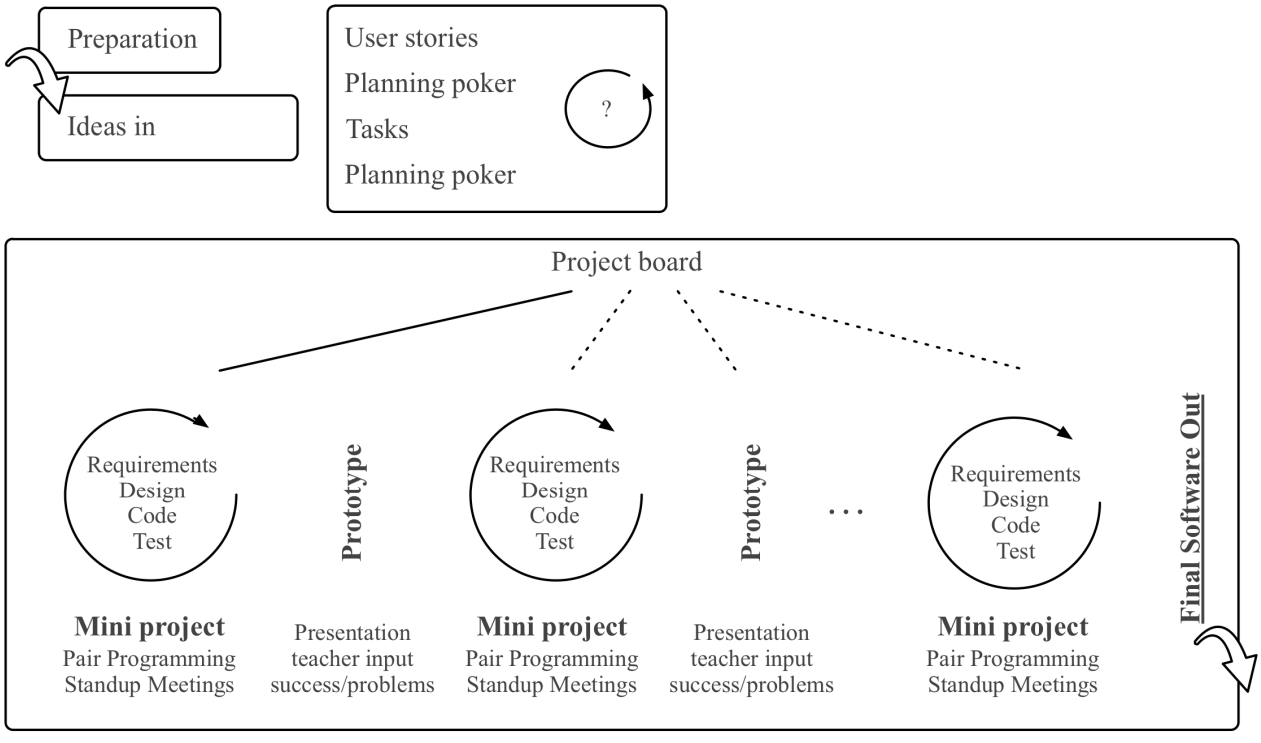

Fig. 7. Agile Model for Projects in Computing Education (AMoPCE).

functionality when there is a more straightforward solution available that passes all automatic tests. Emphasizing this aspect may raise students' awareness of efficiency and for evaluating different solutions.

3.3.3 Keep It Simple

This claim refers to code minimalism: each function should be solved by minimal and straightforward code snippets to ensure an elegant and readable code that is easy to maintain.

4. DISCUSSION

The proposed agile model for SD projects AMoPCE addresses a majority of the previously identified problems. Agile practices fill the learners' gap between requirements and outcome by providing clearly defined strategies for handling difficult planning tasks. Based on the perception of PBL as a team activity, which is in line with modern SD, a team size of 4-6 students is recommended. Dividing the class into several teams, in comparison to having the full class working on one project as proposed e.g. by Schubert and Schwill [39], allows for addressing a broader range of topics, hence it is easier to meet the interest of more students. Also, it should be much easier to find agreement within smaller teams.

Adopting an iterative project design matches the formal circumstances of school projects. In most cases, projects will be worked at along the regular school timetable, sometimes in a so-called project week on several days. In both settings, pedagogical aspects of working in group settings, such as giving curricular input, intermediate project presentations or discussing of common problem solving strategies, can be taken into account easier, since meetings in plenum are part of the model. Correspondingly, projects are divided into mini-projects, which are easier to overview, plan and understand; bureaucratic overhead is reduced. Classroom-management aspects are addressed with professional practices such as standup meetings. This includes recalling of the project status at the beginning of each lesson and quickly planning the individual and group activities for the day.

Within projects, ideas, students' motivation, and skills change over time. Due to the limited time available to work on the projects per week, this is especially very likely for school software projects. Agile methods welcome changes and provide mechanisms to adapt to them with often changing tasks and a straightforward implementation of PBL. Students' confidence is addressed by increasing familiarity stemming from the iterative character of the process.

In comparison to the ACMM, which provides solutions for teachers' difficulties that are grounded in teaching situations, the proposed agile model provides solutions for students' difficulties in learning situations. This in return is expected to relieve the teacher. Giving students clearly defined practices to manage their development processes allows teachers to focus on supporting elaboration and implementation of students' ideas, thus changing the teacher's role from instructor to coach. This better meets the demands of PBL. Additionally, it allows teachers to highlight creativity and social aspects that are rarely seen in connection with computer science (cp. [8; 29]). Applying an adapted SD methodology in school that is also implemented by well known large scale companies may help teachers and students to build an adequate and appealing understanding of computer science relying on creativity, dynamic change, feedback, and soft skills. These attributes may support an attractive notion of computer science, as found in our first experimental settings with agile methods in school projects [18] .

In summary, AMoPCE is suited for supporting the objectives of PBL, for maintaining a professional orientation and for easing the mentoring of software projects. However, in this model curricular

aspects such as content and size of a project are not explicitly considered. Nevertheless, it provides the flexibility to fit into a variety of possible scenarios. Evaluation and assessment aspects, e.g. assessing individual achievement in comparison to group achievement are not determined by the model. However, again practices of SD appear to be adaptable and can be considered: Frequently, agile development teams use self assessments and subsequent peer reviews to verify individual workload and commitment.

As outlined in section 3, there are further practices of agile methods that focus on the quality of the outcome: test driven development, collective code ownership, refactoring, and keep it simple. We acknowledge these practices to be also useful in educational settings but we understand them to be highly dependant on features and methodologies of the used programming environments and tools (e.g. code repositories, automatic testing environments).

The use of agile practices in school SD projects has the potential for replacing the so far predominantly used sequential model. From discussions with teachers we know of the high interest, but a lack of knowledge and resources concerning the use of agile methods. This includes the demand for a revised project model. With AMoPCE, as outlined in this article, we present a model which explains ideas and realization possibilities of agile practices and which can be used as blueprint. In a next step of our research the model will be applied in classroom settings and it will be investigated, to which extend the expected benefits and problem solutions will be approved in practice.

5. REFERENCES

[1] Barak, M., Waks, S., and Doppelt, Y. 2000. Majoring in technology studies at high school and fostering learning. *Learning Environments Research 3*, 2, 135-158.

[2] Beck, K. and Andres, C. 2004. Extreme Programming Explained: Embrace Change (2nd Edition).

[3] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., and Jeffries, R. 2001. Manifesto for agile software development. *The Agile Alliance*, 2002-2004.

[4] Bergin, J. 2000. The Object Game. An Exercise for Studying Objects. *Online at: http://csis.pace.edu/~bergin/patterns/objectgame.html (visited 01.07.2012)*.

[5] Blumenfeld, P.C., Soloway, E., Marx, R.W., Krajcik, J.S., Guzdial, M., and Palincsar, A. 1991. Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. *Educational Psychologist 26*, 369-398.

[6] Blumenfeld, P.C., Soloway, E., Marx, R.W., Krajcik, J.S., Guzdial, M., and Palincsar, A. 1991. Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational Psychologist 26*, 3-4, 369-398.

[7] Braught, G., Wahls, T., and Eby, L.M. 2011. The Case for Pair Programming in the Computer Science Classroom. *Transactions on Computing Education 11*, 2:1--2:21.

[8] Caspersen, M.E. and Kölling, M. 2009. STREAM: A First Programming Process. *Transactions on Computing Education 9*, 4:1--4:29.

[9] Chevallard, Y. 1988. On didactic transposition theory: Some introductory notes. In *International Symposium on Research and Development in Mathematics, Bratislava, Czechoslavakia*.

[10] Christopher, G.J. 2004. Test-driven development goes to school. *J. Comput. Small Coll. 20*, 1, 220-231.

[11] Dewey, J. and Kilpatrick, W.H. 1935. *Der Projekt-Plan: Grundlegung und Praxis*. Hermann Böhlaus Nachfolger.

[12] Diethelm, I. 2007. Strictly models and objects first - Unterrichtskonzept und -methodik für objektorientierte Modellierung im Informatikunterricht Universität Kassel, Kassel.

[13] Fincher, S. and Petre, M. 1998. Project-based learning practices in computer science education. In *IEEE Frontiers in Education Conference (Tempe, Arizona)*, 1185-1191.

[14] Fothe, M. 2007. Algorithmen in spielerischer Form. In *Praxisband der GI-Fachtagung Informatik und Schule*, 31-42.

[15] Frey, K. 1983. Die sieben Komponenten der Projektmethode - mit Beispielen aus dem Schulfach Informatik. *LOG IN 3*, 2, 16-20.

[16] Frey, K. and Schäfer, U. 1982. *Die Projektmethode*. Beltz, Weinheim [a.o.]

[17] Gal-Ezer, J., Beeri, C., Harel, D., and Yehudai, A. 1995. A high school program in computer science. *Computer 28*, 10, 73-80.

[18] Göttel, T. 2012. *The image of Computer Science and social aspects of modern software development processes in school contexts*. University of Hamburg, Hamburg.

[19] Hartmann, W., Näf, M., and Reichert, R. 2006. *Informatikunterricht planen und durchführen*. Springer.

[20] Hazzan, O. and Dubinsky, Y. 2007. Why Software Engineering Programs Should Teach Agile Software Development. *SIGSOFT Software Engineering Notes 32*, 1-3.

[21] Hazzan, O., Dubinsky, Y., and Meerbaum-Salant, O. 2010. Didactic transposition in computer science education. *ACM Inroads 1*, 4, 33-37.

[22] Highsmith, J. and Cockburn, A. 2001. Agile Software Development: The Business of Innovation. *Computer 34*, 120-122.

[23] Humbert, L. 2005. *Didaktik der Informatik*. Teubner.

[24] Koerber, B. 1992. Die Angst des Lehrers vorm Projektunterricht. *LOG IN 12*, 5/6, 3.

[25] Krajcik, J.S., Czerniak, C., and Berger, C. 1999. *Teaching children science: A project-based approach*. McGraw-Hill Boston, MA.

[26] Larman, C. and Basili, V.R. 2003. Iterative and Incremental Development: A Brief History. *Computer 36*, 47-56.

[27] Layman, L., Williams, L., and Cunningham, L. 2004. Exploring Extreme Programming in Context: An Industrial Case Study. In *Proceedings of the Agile Development Conference* IEEE Computer Society, Washington, DC, USA, 32-41.

[28] Loftus, C. and Ratcliffe, M. 2005. Extreme Programming Promotes Extreme Learning? In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* ACM, New York, NY, USA, 311-315.

[29] Maass, S. and Wiesner, H. 2006. Programmieren, Mathe und ein bisschen Hardware... Wen lockt dies Bild der Informatik? *Informatik-Spektrum 29*, 2, 125-132.[30] Magenheim, J., Nelles, W., Rhode, T., Schaper, N., Schubert, S., and Stechert, P. Competencies for informatics systems and modeling: Results of qualitative content analysis of expert interviews IEEE, 513-521.

[31] Mann, C. and Maurer, F. 2005. A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction. In *Proceedings of the Agile Development Conference* IEEE Computer Society, Washington, DC, USA, 70-79.

[32] Meerbaum-Salant, O. and Hazzan, O. 2009. Challenges in Mentoring Software Development Projects in the High School: Analysis According to Shulman. *Journal of Computers in Mathematics and Science Teaching 28*, 1, 21.

[33] Meerbaum-Salant, O. and Hazzan, O. 2010. An Agile Constructionist Mentoring Methodology for Software Projects in the High School. *Transactions on Computing Education 9*, 21:21--21:29.

[34] Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., and Balik, S. 2003. Improving the CS1 Experience with Pair Programming. *ACM SIGCSE Bulletin 35*, 359-362.

[35] Papert, S. and Harel, I. 1991. Situating Constructionism. In *Constructionism*, S. PAPERT and I. HAREL Eds. Ablex Publishing, Norwood, N.J.

[36] Pilone, D. and Miles, R. 2008. *Head first software development*. O'Reilly Media, Inc.

[37] Rico, D.F. and Sayani, H.H. 2009. Use of Agile Methods in Software Engineering Education. In *Proceedings of the 2009 Agile Conference* IEEE Computer Society, Washington, DC, USA, 174-179.

[38] Schneider, J.-G. and Johnston, L. 2005. eXtreme Programming: Helpful or Harmful in Educating Undergraduates? *Journal of Systems and Software 74*, 121-132.

[39] Schubert, S. and Schwill, A. 2011. *Didaktik der Informatik*. Springer.

[40] Schwaber, K. and Beedle, M. 2001. Agile Software Development with Scrum.

[41] Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., and Verno, A. 2003. A model curriculum for K-12 computer science: Final report of the ACM K-12 task force curriculum committee. *New York, NY: The Association for Computing Machinery*.

[42] Weigend, M. 2005. Extreme Programming im Klassenraum. In *INFOS 2005 - 11. GI-Fachtagung Informatik und Schule* S. FRIEDRICH Ed. GI-Edition - Lecture Notes in Informatics (LNI), Dresden, Germany.

Entwicklung eines agilen Frameworks für Projektunterricht mit Design-Based Research

Petra Kastl und Ralf Romeike[1]

Abstract: Fachdidaktische Innovationen stellen Forscher und Praktiker grundsätzlich vor die Herausforderung, Theorie mit Praxis in Einklang zu bringen. Besonders in der Informatik bergen kontinuierliche fachliche Weiterentwicklungen bedeutendes Potential für didaktische und methodische Neuerungen. Der Beitrag skizziert am Beispiel eines agilen Modells für Projekte einen Forschungsprozess, der die Implementierung und Weiterentwicklung des Modells unter Einbeziehung unterrichtspraktischer Expertise begleitet. Zusätzlich werden Erfahrungen aus der ersten Iteration des dem Prozess zugrunde liegenden Design-Based Research-Ansatzes beschrieben.

Keywords: agile Methoden, agile Praktiken, Design-Based Research, Projektunterricht

1 Motivation

In der professionellen Softwareentwicklung hat sich in den letzten Jahren herausgestellt, dass sequentielle Vorgehensmodelle wie das Wasserfallmodell oft zu mangelhafter Qualität, größeren Terminverschiebungen und wenig Kundenzufriedenheit führen – für Entwickler eine frustrierende Situation. Ähnliche Probleme treten auch in Schulprojekten auf, die sich am Wasserfallmodell orientieren: Projekte sind schwer planbar, werden aus Zeitmangel oft nicht mit einem nutzbaren Produkt beendet und lange Modellierungs- und Testphasen sind kaum zu motivieren [HNR07, We05]. In der IT Branche setzt man zur Lösung mehr und mehr auf agile Methoden, also verstärkt auf Kommunikation und Kooperation, Mitarbeitermotivation sowie eine kontinuierliche Produkterstellung in kurzen, iterativen Entwicklungsphasen, die durch eine langfristige Produktvision gelenkt werden. Agile Methoden beruhen entsprechend des agilen Manifests [Be01] auf Werten und Praktiken, die auch für den Schuleinsatz angebracht erscheinen. Auf der WiPSCE 2012 stellten Romeike und Göttel ein auf agilen Methoden basierendes Modell für Projekte im Informatikunterricht (AMoPCE) vor [RG12]. Das Modell stellt für die Praxis einen leicht einsetzbaren Satz agiler Praktiken bereit, der es erlaubt im Unterricht agile, kommunikationsfördernde Prozesse einzusetzen, die schnell nutzbare Teilergebnisse liefern und die das Entwicklerteam ins Zentrum stellen.

Ein Problem bei der theoretischen Entwicklung eines so umfangreichen Modells ist, dass praxisrelevante Aspekte und auftretende Schwierigkeiten nur eingeschränkt vorhergesehen werden können. Um Schwierigkeiten zu vermeiden, wie sie bei der Umsetzung von

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Didakitk der Informatik, Martensstr. 3, 91058 Erlangen petra.kastl@fau.de, ralf.romeike@fau.de

am Wasserfallmodell orientierten, linearen Modellen im Schulunterricht auftreten, soll das theoretisch entwickelte agile Modell in verschiedenen schulpraktischen Kontexten erprobt und mit Hilfe eines Design-Based Research-Ansatzes in einem zyklischen Prozess verfeinert und weiterentwickelt werden. Ziel ist es, die berufsspezifischen Kompetenzen erfahrener Lehrkräfte in den Prozess einzubinden und ihre Beobachtungen bei der Umsetzung, ihre Ideen, Interpretationen und individuellen Anpassungen in jedem Zyklus zu bündeln und zu reflektieren, um die Weiterentwicklung mittels formativer Evaluation empirisch abzusichern. Schrittweise und forschungsgeleitet soll AMoPCE so zu einem praxiserprobten, offenen und flexiblen „Methodenkoffer" weiterentwickelt werden, mit dessen Hilfe Projekte mit einer individuell angepassten, sinnhaften agilen Vorgehensweise erfolgreich durchgeführt werden können. Das vorgestellte Vorgehen und die zugrundeliegenden Überlegungen können als Basis dienen, auch andere Innovationen forschungsgeleitet in der Praxis des Informatikunterrichts zu verankern.

2 Projekte im Informatikunterricht

Projektunterricht beschreibt unabhängig vom Schulfach eine ganzheitliche, offene Lernform, die mit hohem Anteil an Mitbestimmung der Teilnehmer in Phasen abläuft [FS82]. Beschrieben wird er meist über eine Liste von Arbeitsschritten und/oder Merkmalen, wie beispielsweise bei Gudjons oder Frey. Ausgehend von komplexen Aufgaben oder offenen Fragestellungen entwickeln und konkretisieren die Teilnehmer entsprechend ihren Neigungen und Interessen Ideen, bei deren Umsetzung sie selbstorganisiert und zielgerichtet arbeiten und die Verantwortung für das Projekt gemeinsam übernehmen [Gu14]. Sie benötigen dazu vielfältige fachliche, methodische und soziale Fähigkeiten und Fertigkeiten, deren Erwerb, Anwendung und Weiterentwicklung Ziele von Projektunterricht sind [Fr83]. Eine bekannte Schwierigkeit ist, dass Schülerinnen und Schüler (SuS) mit der Komplexität und den Anforderungen eines Projekts oft überfordert sind. Aufgabe der Lehrkraft ist es dann, das Projekt in kleinere Teilprojekte zu gliedern [SS05].

Bei Softwareentwicklungsprojekten im Informatikunterricht liegt eine besondere Situation vor: Da professionelle Softwareentwicklung in Projekten stattfindet, gibt es hierfür seit den 1970iger Jahren wissenschaftlich verankerte Vorgehensmodelle. Will man den Lernenden einen Einblick in die „echte Welt" vermitteln, muss man diese Modelle berücksichtigen. Gleichzeitig stehen sie zum Teil im Widerspruch zu wesentlichen Kriterien und Zielen von Projektunterricht. Daher findet man – historisch bedingt – in der Informatik überwiegend adaptierte Modelle, die versuchen, die Kernideen und Ziele von Projektunterricht bestmöglich mit dem Wasserfallmodell zu verweben [Fr83, SS11]. Unterhält man sich mit Lehrkräften über Projekte und studiert man didaktische Literatur, legt das die Vermutung nahe, dass solche linearen Modelle aufgrund schulischer Anforderungen und Organisationsstrukturen in der Umsetzung häufig zu vielerlei Problemen führen [HNR07, Hu05, MH10, SS11, We05]. SuS sind regelmäßig mit der Komplexität überfordert und können Inhalte und Ressourcen kaum selbständig organisieren bzw. planen. Eine Folge ist, dass sie die Verantwortung für das Projekt nicht gemeinsam

übernehmen können, sodass oft geringe Eigeninitiative und Motivation bei Teilen des Teams beobachtet wird[2]. Gleichzeitig muss die Lehrkraft in hohem Maße motivieren, unterstützen, planen und leiten[3,4]. Aber auch für Lehrkräfte ist die Planung innerhalb der Organisationsstrukturen von Schule schwierig. Unfertige Produkte und schlechte Produktqualität sind die Folge[5,6]. Diese können dann auch nur sehr eingeschränkt von SuS reflektiert und bewertet werden. Eine weitere Schwierigkeit für Lehrer besteht darin, dass das Wasserfallmodell keine konkreten Techniken zur Verfügung stellt, die Kommunikation und Interaktion unterstützen, um soziales Lernen zu ermöglichen.

Theoretische Überlegungen zu agilen Methoden lassen vermuten, dass sie den Spagat zwischen den Zielen professioneller Softwareentwicklung (effiziente Erstellung nützlicher Software von hoher Qualität zur Zufriedenheit des Kunden) und den oben dargestellten Zielen von Schulprojekten müheloser schaffen als lineare Modelle, weist man ihnen doch Eigenschaften wie verstärkte Betonung von Kommunikation und Kooperation sowie individueller Fähigkeiten und Bedürfnisse zu, ebenso die Stärkung der Eigenverantwortung und das Setzen auf motivierte Projektbeteiligte, die von sich aus verantwortungsvoll und couragiert handeln [Be01, Ru04]. Von einem agilen Modell für Schulprojekte könnten demnach Lernende und Lehrende gleichermaßen profitieren zumal eine an das individuelle Projekt angepasste „schlanke“ Vorgehensweise sinnhaft, zeitgemäß und realitätsnah ist. Vereinzelte Untersuchungen in der Vergangenheit bestärken diese Annahmen: Weigend [We05] beschreibt positive erste Erfahrungen bei der Anwendung einzelner agiler Elemente. Meerbaum-Salant und Hazzan [MH10] haben eine Studie durchgeführt, in der sie agile Werte nutzen, um Lehrer bei der Betreuung von Softwareprojekten zu unterstützen. Göttel [Gö12] verwendet einige agile Praktiken in verschiedenen Projekten mit der Intention, soziale Interaktionen in der modernen Softwareentwicklung hervorzuheben und so Lernende ein attraktives Bild der Informatik zu vermitteln.

3 Das agile Framework AMoPCE

Agile Methoden stellen konkrete Vorgehensweisen für die Projektorganisation und Projektarbeit zur Verfügung, die flexibel und nach den individuellen Bedürfnissen eines Projekts ausgewählt werden. Wesentliche innovative Neuerungen gegenüber dem linearen Modell werden im Folgenden kurz dargestellt[7].

[2] „Einer hat dann zu Hause den größten Teil programmiert.“ (Interview mit einem Lehrer)
[3] „In jedem Kurs das Gleiche: Die Leute wollen einfach nicht zuhören, wenn es darum geht, zuerst die Struktur einer Website auf Papier zu überlegen.“ [HNR07]
[4] „Obwohl ich nach den Doppelstunden regelmäßig nass geschwitzt war, haben die Schülerinnen und Schüler vor allem die langen Wartezeiten bemängelt, bis Unterstützung kam.“ (Interview mit einem Lehrer)
[5] „Zum Testen sind wir nicht mehr gekommen.“ (Interview mit einem Lehrer)
[6] „Ein paar Fehler sind noch drin, so dass es noch nicht läuft. Aber zum Fertigstellen hatten wir keine Zeit mehr.“ (Interview mit einem Lehrer)
[7] Eine detaillierte Beschreibung insbesondere auch der Anpassung professioneller Elemente an den Schulkontext durch didaktischen Transposition findet man bei Romeike und Göttel [RG12].

Iteratives Vorgehen und Prototypen: Ausgehend von einer Produktvision werden die gewünschten Funktionalitäten in User Stories aus Nutzersicht beschrieben und priorisiert. In den anschließenden Iterationen werden jeweils einige User Stories mit hoher Priorität ausgewählt und die zugehörigen Tasks formuliert, die aus Entwicklersicht bei der Umsetzung der jeweiligen Story zu erledigen sind. In einem (linearen) Mini-Projekt mit Planungs-, Entwurfs- Implementierungs- und Testphase werden dann die Tasks realisiert. Am Ende eines jeden Mini-Projekts steht ein lauffähiger und getesteter Prototyp, der ausprobiert, vorgestellt und bewertet werden kann (vgl. Abb. 1).

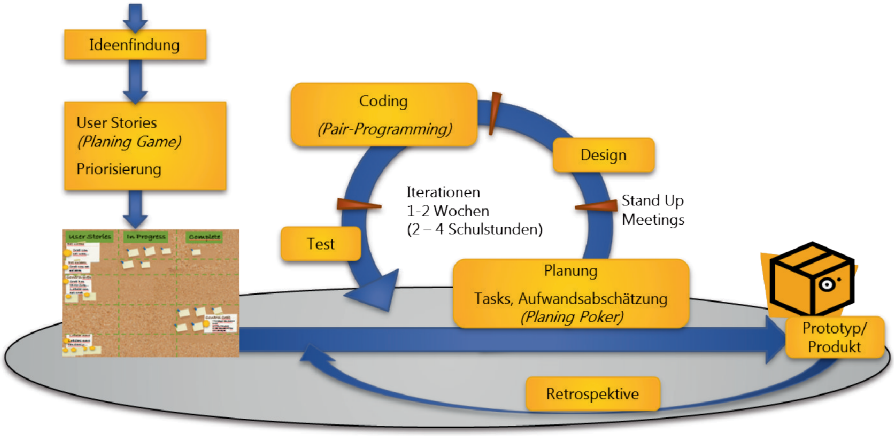

Abb. 1: Ein agiles Framework für Schulprojekte

Solche Mini-Projekte sind wesentlich leichter zu überblicken und sollen den Schülerinnen und Schülern das autonome Planen und Arbeiten erleichtern. Durch die iterative Vorgehensweise können die Projektteams auch mit geringer fachlicher und methodischer Kompetenz beginnen, und Erkenntnisse, die sie durch Reflexion und Peerfeedback gewinnen, in der nächsten Iteration anwenden. Der Abschluss jeder Phase mit einem lauffähigen Prototyp führt zu sichtbaren Ergebnissen, die die Motivation erhalten und erleichtert gleichzeitig die Gesamtplanung durch die Lehrkraft.

Anforderungen als variable Projektgröße: Während in professionellen agilen Projekten in der Regel Zeit, Personal und Produktqualität fixe Größen darstellen, sind Anforderungen flexibel, folgen einer Produktvision und werden durch intensive Zusammenarbeit mit dem Kunden nach und nach konkretisiert. Der Vorgang der Softwareentwicklung wandelt sich: Aus einem stark prozessfokussierten Vorgang wird ein dynamischer und kreativer [FST13], der die Projektbeteiligten zu gestalterischem und verantwortungsvollem Handeln ermutigt. Reichen die Ressourcen nicht aus, werden keine Kompromisse hinsichtlich Qualität (Testen), Zeit (Endtermin) oder mehr Personal eingegangen, sondern niedrig-priorisierte Anforderungen nicht umgesetzt. Diese innovative Herangehensweise erleichtert die Planungsarbeit in Schulprojekten enorm, weil sie Planungsfehler unerfahrener Schüler verzeiht. Darüber hinaus kann die Lehrkraft die Rolle des Kun-

den übernehmen und damit steuernd eingreifen.

Kommunikationsfördernde Praktiken: Einige Praktiken wie Pair-Programming und Stand-Up Meetings, in denen sich die Teammitglieder gegenseitig kurz über Erledigtes, Probleme und Geplantes informieren, fördern die Kommunikation und geben der Lehrkraft einen Einblick in den individuellen Lernfortschritt.

Praktiken, die den Planungs- und Organisationsprozess unterstützen: Durch Praktiken wie das Planning Poker erhalten SuS Handlungsanweisungen, die sie in der schwierigen Phase der Aufwandsabschätzung unterstützen sollen. Regelmäßige Stand-Up Meetings, feste Iterationslängen und Retrospektiven am Ende jeder Iteration strukturieren den Prozess. Das Project Board ist zentraler Informations- und Planungsbereich, an dem der Projektfortschritt visualisiert und begreifbar gemacht wird und auftretende Probleme und Fragen notiert werden können. Es soll Lernenden und Lehrenden einen Überblick über den Stand des Projekts bieten, die Planung der nächsten Schritte erleichtern und helfen, Entscheidungen zu treffen. Im Zusammenspiel sollen diese Praktiken Schülerinnen und Schülern ein selbständiges Arbeiten und Planen ermöglichen und in Konsequenz eine starke Änderung der Lehreraufgaben bewirken.

Die theoretischen Überlegungen zeigen, dass ein agiles Framework Projektarbeit für Lernende und Lehrende im positiven Sinne verändern kann. Neben der begründeten Relevanz und der Konsistenz gilt es in den nächsten Schritten die Praxistauglichkeit sicher zu stellen, um dann – möglichst umfassend und gestützt durch theoretische und empirische Argumente – die Bedingungen zu beschreiben, die zum Gelingen führen. Ein dazu geeignetes Forschungsformat erlaubt es, Lern- und Arbeitsprozesse während des Projektverlaufs bezüglich hinderlicher und fördernder Faktoren zu untersuchen und liefert idealerweise neben dem Erkenntnisgewinn für die Theorie auch ein für die Praxis nützliches Artefakt, das zukünftig eine Verbesserung der Unterrichtspraxis erleichtert.

4 Forschungsgeleitete Adaption und Weiterentwicklung

Als Problem der Unterrichtsentwicklung hat sich gezeigt, dass allein durch theoretische Überlegungen erlangte Erkenntnisse und Modelle oft nur unzureichend zu nachhaltiger Innovation in der Unterrichtspraxis führen [Re05]. Die Gründe hierfür sind vielfältig. So können beispielsweise verschiedenste Probleme bei der Implementierung eines theoretisch erarbeiteten Modells in realen Kontexten auftreten, wie beim Verwenden des adaptierten Wasserfallmodells. Diese können inhärent dem Entwurf innewohnen oder abhängig vom Kontext auftreten. Manchmal fehlt rein theoretisch begründeten Ansätzen auch einfach die Praxisrelevanz. Gleichzeitig ist zeitgemäßer Informatikunterricht auf kontinuierliche didaktische Aufarbeitung der Themen und auf nachhaltige Innovation in der Unterrichtspraxis angewiesen, da Informatik eine dynamische Wissenschaft ist. Beispiele hierfür finden sich in der fortwährenden Entwicklung neuer Software- und Hardwarewerkzeuge zur Verbesserung der Unterrichtspraxis oder der Weiterentwicklung zentraler Unterrichtsgegenstände, wie sie z. B. derzeit beim Thema Datenmanagement, -schutz

und -sicherheit zu beobachten ist. Die agilen Methoden schließlich haben den berufstypischen Prozess der Softwareentwicklung ebenso wie die wissenschaftliche Sicht auf ihn enorm verändert und vermutlich haben sie auch das Potential, Unterrichtsprojekte gewinnbringender zu gestalten. Einer engen Theorie-Praxis-Verschränkung scheint deshalb in der Informatik eine substantielle Bedeutung inne zu wohnen, damit in der Theorie begründete und erarbeitete Entwürfe forschungsgeleitet so weiterentwickelt werden können, dass sie zu nachhaltiger Innovation und konkreter Unterrichtsweiterentwicklung führen.

Ein vielversprechendes Forschungsformat, das hilft generalisiertes Wissen über das Entwerfen und das nachhaltige Implementieren innovativer Unterrichtsmodelle zu entwickeln, ist Design-Based Research (DBR) [De03]. Reed und Guzdial [RG12] halten DBR für die Informatikdidaktik für besonders geeignet, da hiermit Polarisierungen und Diskussionen des „einzigen richtigen Vorgehens“ vermieden werden können und stattdessen Interventionen hinsichtlich der Ursachen erfolgreicher Lernprozesse beschrieben werden. Charakteristische Merkmale des DBR sind die Motivation, Unterrichtspraxis verbessern zu wollen, das Ziel, reale Probleme zu lösen und dabei Erkenntnisse für Theorie und Praxis zu gewinnen, sowie der Stellenwert des Designs als zentraler Teil des Forschungsprozesses. DBR ermöglicht es, verschiedenste Aspekte einer Intervention in unterschiedlichen Kontexten zu untersuchen, um Designprinzipien zu generieren, die durch formative Evaluation der Beobachtungen und Erfahrungen empirisch gestützt sind [CJB04]. Die Arbeit verläuft typischerweise zyklisch und zusammen mit Praktikern, wobei der Entwurf der Intervention flexibel ist – sowohl innerhalb eines Zyklus als auch über den Forschungsprozess hinweg. Deshalb ist insbesondere bei der Analyse und der Reflexion der Beobachtungen und beim Re-Design ein intensiver Austausch zwischen Forschern und Lehrerinnen und Lehrern wichtig.

Aus den oben beschriebenen Gründen sind wir der Meinung, dass die Theorie-Praxis-Verschränkung und eine Zusammenarbeit von Forschern und Lehrkräften auf Augenhöhe für unsere Arbeit besonders wichtig sind. Das Forschungsvorhaben fußt deshalb auf der berufsspezifischen Kompetenz von Lehrkräften, fachliches Wissen, allgemeindidaktisches, fachdidaktisches sowie pädagogisches Wissen in einem Wissensbereich zu integrieren [St10]. Erfahrene Lehrkräfte passen das agile Modell individuell an ihren Kontext an, setzten ergänzend eigene Ideen um, beobachten und sammeln Erfahrungen.

Der DBR-Prozess zu den agilen Methoden gliedert sich in Phasen des Inputs und der Vernetzung (Workshops) sowie Phasen der individuellen Anpassung, Weiterentwicklung und Erprobung durch die Lehrkräfte (vgl. Abb. 2). Wichtigste Funktion der Workshops ist die Bündelung der Beobachtungen und Erfahrungen aus den Implementierungen, um in gemeinsamer Analyse und Reflexion die Erkenntnisse herauszuarbeiten, auf deren Basis die Theorie angepasst und Lösungsansätze und Anregungen für die weitere Verbesserung der Praxis entwickelt werden. Eine Datenerhebung findet in jedem Zyklus statt und umfasst Leitfadeninterviews, exemplarische Projektdokumentationen und -ergebnisse sowie die Ergebnisse des Workshops. Ziel ist es, die Interventionen als Fallstudien bezüglich möglichst vieler beeinflussender Faktoren zu rekonstruieren und die

theoretischen Überlegungen, die Ergebnisse aus den Workshops und die Beobachtungen in den Interventionen auf Konsistenz zu prüfen.

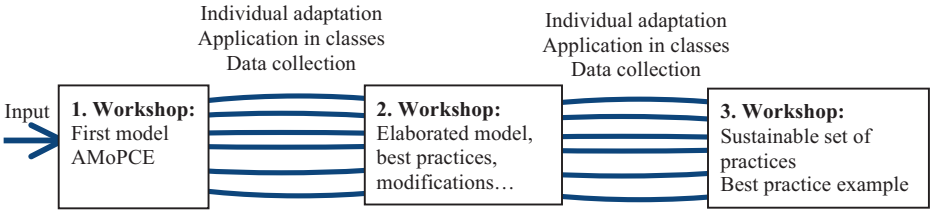

Abb. 2: Forschungsprozess zur Weiterentwicklung von AMoPCE

5 Durchführung des ersten Zyklus und erste Ergebnisse

Im Folgenden beschreiben wir wesentliche Teile der konkreten Umsetzung des ersten Zyklus, insbesondere mit Blick auf die Ausgestaltung der Zusammenarbeit mit den Lehrkräften, eine Einordnung in den Gesamtkontext sowie erste Erkenntnisse.

Die Idee einer Zusammenarbeit von Theoretikern und Praktikern auf Augenhöhe und die aktive Einbeziehung des berufsspezifischen Wissens der Lehrkräfte von Beginn an waren entscheidend für die Gestaltung des ersten Workshops, der zugleich die Funktion einer Fortbildung haben musste, damit die Lehrkräfte nach der Teilnahme in der Lage waren, das agile Modell zu implementieren. Interessierte Lehrkräfte sollten dazu ermutigt werden, das agile Modell in einer an ihren Kontext angepassten Form in einem Unterrichtsprojekt anzuwenden und dabei auch eigene Ideen und Interpretationen einzubauen. Darüber hinaus sollten sie motiviert werden, ihr berufsspezifisches Wissen in einen theorie-praxis-verschränkten Prozess der Weiterentwicklung einzubringen. Am ersten, zweitägigen, Workshop, nahmen elf Lehrerinnen und Lehrer sowie drei Forscher teil. Zur Motivierung, Verdeutlichung der Praxisrelevanz und thematischen Einstimmung beschrieb ein Softwareentwickler zunächst an einem konkreten Projekt, wie in seinem Team professionelle agile Softwareentwicklung umgesetzt wird und begründete deren Sinnhaftigkeit an konkreten, eingängigen Beispielen. Im Anschluss wurden typische Probleme in Schulsoftwareprojekten besprochen, AMoPCE vorgestellt und Möglichkeiten praktischer Umsetzungen der einzelnen Vorgehensweisen des Modells ausführlich und offen diskutiert. In dieser Phase haben die Lehrkräfte aktiv mögliche Schwierigkeiten und Probleme herausgearbeitet, die bei der Implementierung von AMoPCE im Klassenraum auftreten können und erste Ideen zu deren Lösung entwickelt. Nach einer anschließenden praktischen Erprobung wurden zu allen Vorgehensweisen des agilen Modells die diskutierten Punkte fixiert sowie weitere Ideen und Lösungsansätze dazu erarbeitet. Im weiteren Verlauf des ersten Zyklus haben sechs Lehrerinnen und Lehrer aus drei Bundesländern die agilen Methoden in sechs Mittelstufenklassen und drei Oberstufenkursen mit insgesamt ca. 170 SuS umgesetzt. Als Werkzeuge wurden Scratch, Greenfoot, BlueJ und eine Processing-IDE eingesetzt, die Projektlänge variierte von wenigen

Wochen bis hin zu acht Monaten und ebenso unterschiedlich waren die Vorkenntnisse der SuS, die vom Programmieranfänger bis zum fortgeschrittenen Programmierer reichten. Vorrangiges Ziel der ersten Implementierungsphase war es, die Stellen im Prozess zu identifizieren, die den SuS bzw. den Lehrkräften noch generell oder kontextbedingt Probleme bereiten. Ein weiteres Ziel war es herauszufinden, inwiefern sich die theoretischen Überlegungen in der Praxis an Stellen zeigen, an denen die Umsetzung bereits ohne Schwierigkeiten verlief und welche Kontextfaktoren dabei eine Rolle spielen. Auch hier galt es, die Perspektiven der Lernenden und die der Lehrenden zu berücksichtigen. Die Interpretationen, Ideen und Anpassungen der Lehrer stellen einen wesentlichen Schritt hin zu einem an Best Practices orientierten Leitfaden dar, der Lehrer bei Kontextanpassungen unterstützen soll. Diese Aspekte wurden in Leitfadeninterviews erfasst.

Generell zeigte sich in den Interviews, dass die Erfahrungen und Beobachtungen unabhängig vom Kontext motivierend gut mit den theoretischen Überlegungen übereinstimmten. Klare Schwierigkeiten zeigten sich lediglich bei SuS mit wenig Implementierungserfahrung die Java als Programmiersprache verwendeten. Diesen SuS fiel das Formulieren von Tasks schwer, also der Perspektivwechsel vom Nutzer zum Entwickler. Abgesehen von diesem Hemmnis berichten die Lehrer übereinstimmend, dass die Teams mit Hilfe der agilen Praktiken ihre Projektarbeit von Beginn an selbst organisiert haben. So berichtet ein Lehrer zwar von anfänglichen Schwierigkeiten einer Gruppe, die „innerhalb von den ersten 45 Minuten des Unterrichts fünf Stand-Up Meetings brauchte“ aber auch, dass sie rasch lernten „zielgerichtet [zu] planen“. Ein anderer formuliert den Vorteil für SuS so: „Sie haben halt eine Struktur, in der sie ihre Fehler, ihre Probleme lösen können. Ich fand […], dass sie auch mehr Selbständigkeit erlebt haben.“ Weil sie ihnen einen regelmäßigen, guten Einblick in die fachlichen und sozialen Fähigkeiten der SuS boten, schätzten die Lehrer die Stand-Up Meetings auch für sich als sehr wertvoll ein. So berichtet ein Lehrer „ich lerne auch was über die, ja, wie denken die, wie ticken die“ und weiter führt er aus „du siehst Entwicklungen, wo ich der Meinung bin, die hätte ich sonst eben nicht gesehen“ und fügt hinzufügt „ich hab viel mehr Zeit auch, die Schüler zu beobachten“, womit er die geänderte Lehrerrolle anspricht. „Ich fand es total interessant,“ führt ein anderer Lehrer aus, „…also du hängst nicht mitten drin, du musst nicht moderieren, sondern du kannst einfach zuhören. Klar, man kriegt sehr viel mehr mit, darüber wie sie arbeiten, wie sie denken, wie dieser Prozess abläuft.“ Im Zusammenspiel mit dem iterativen Vorgehen konnten sie die Lernfortschritte der SuS erkennen: „Also die Kommunikation und die Kooperation ist von Mal zu Mal intensiver geworden, also von Iteration zu Iteration.“ Sagt ein Lehrer und fügt erklärend an, dass sich immer mehr SuS an fachlich immer tiefergehenden Gesprächen beteiligten: „Wenn du so zuhörst, was da an Problemerörterungen nach dem gegenseitigen Testen lief, ist [das] beeindruckend.“ Auch der organisatorische Aspekt wurde durchweg als hilfreich empfunden. Dazu führt ein Lehrer aus: „[Alle] zusammen gekommen sind wir immer am Ende der Stunde […], da gab es immer diese Vorstellung der Prototypen“. Testen wurde nach Aussagen der Lehrerinnen und Lehrer zu einem normalen Bestandteil der Projektarbeit, der uneingeschränkt positiv von den SuS aufgenommen wurde und keiner weiteren Motivierung bedurfte: „Dann waren die gegenseitigen Tests immer erst – rein aus Motivation: Ich will mal sehen, was die gemacht haben“ berichtet ein Lehrer und kontrastiert die

aktuelle Situation etwas später zu Erfahrungen mit dem Wasserfallmodell; „weil wenn sie überhaupt am Testen sind, ist ja das schon mal was." Ein Lehrer berichtet, dass eine Gruppe von sich aus typische, wiederkehrende Fehler identifizierte „und diese Fälle haben sie sich […] auch aufgeschrieben, auch dokumentiert, welche Fehler es sind. Und die bei jeder Iteration wieder getestet. Also dann schon so ein bisschen Regressionstest". Die Prototypen gaben Gelegenheit zu (Peer-)Feedback, so schwärmt ein Lehrer, dass „dieser Kurs von mir permanent gelobt wurde, was für Schüler ich da habe […] weil die haben sich wirklich auch Mühe gegeben, […] tolle Sachen gebracht".

6 Diskussion

Das Gesamtbild aller Interviews, von dem wir oben einen Ausschnitt skizziert haben, zeigt, dass sich die theoretischen Überlegungen in der Unterrichtspraxis unabhängig vom Kontext widerspiegelten. Die Anpassungen an das spezifische Projekt waren durchweg gelungen und werden in Best Practices festgehalten. Interessant war auch, dass die SuS selbst erkannten, welche Praktiken für sie sinnvoll und hilfreich waren. So hat ein Lehrer beispielsweise das Planning Poker erst während des laufenden Projekts eingeführt, als die SuS schon ein Gefühl für sinnvolle Größen von User Stories und Tasks hatten. Diese SuS haben in der Aufwandsabschätzung keinen planerischen Mehrwert gesehen und die Praktik nach zwei Iterationen aufgegeben – gleiches wurde uns auch von professionellen Softwareentwicklern berichtet. In einem anderen Projekt wurde das Abschätzen am Anfang eingeführt und von den SuS als hilfreich empfunden, wobei sie die Erfahrung aus der ersten Iteration nutzten, um im Folgenden vergleichend abzuschätzen.

Interessant ist auch der Aspekt der Nachhaltigkeit. Ein Lehrer wurde von einer Kollegin gefragt, was er mit seinen SuS im Vorjahr gemacht habe. „Die fangen direkt an zu arbeiten, wenn ich eine Aufgabe stelle. Meine heben erst mal alle den Finger." sagte sie. Möglicherweise hat sich die Selbstwirksamkeit und/ oder die Herangehensweise der SuS an komplexere Aufgaben durch das Projekt verändert.

Wir waren überrascht, mit welcher Leidenschaft und mit welchem Engagement die Lehrerinnen und Lehrer sich beteiligen. Dies manifestierte sich beispielsweise in der mutigen und kreativen Umsetzung, die viel Vorbereitungszeit beanspruchte und unter anderem die Idee der „Student Story" hervorbrachte, ein vom Lehrer zu den User Stories der Schüler hinzugefügter Lernauftrag. Auch die inspirierende Atmosphäre und die engagierte Zusammenarbeit in den Workshops zeigt, wie sehr sich die Lehrerinnen und Lehrer mit dem Thema identifizieren. Möglicherweise ist dieses Format auch geeignet, um Lehrerfortbildungen über Zusammenarbeit auf Augenhöhe nachhaltiger zu gestalten, als durch rein inputorientierte Formate. Im Weiteren ist vorgesehen, diesen Aspekt mit zu betrachten, um festzustellen, ob aus der Anlage und Durchführung des Projekts auch Aussagen getroffen werden können, wie theorie-praxis-verschränkte Lehrerfortbildungen effektiv und nachhaltig durchgeführt werden können. Durch den zyklischen Verlauf und das flexible Design ermöglicht es DBR, diese interessanten, unerwartet aufgetreten Aspekte in einem nächsten Zyklus mit zu untersuchen.

Literaturverzeichnis

[Be01] Beck, Kent; Beedle, Mike; Bennekum, Arie van; et al.: Manifesto for Agile Software Development. http://www.agilealliance.org/the-alliance/, abgerufen am 10.04.2015.

[CJB04] Collins, Allan; Joseph, Diana; Bielaczyc, Katerine: Design Research: Theoretical and Methodological Issues. Journal of the Learning Sciences 13 (2004), Nr. 1, S. 15-42.

[De03] Design-based Research Collective: Design-Based Research: An Emerging Paradigm for Educational Inquiry. In: Educational Researcher vol. 32 (2003), Nr. 1, S. 5–8.

[Fr83] Frey, Karl: Die sieben Komponenten der Projektmethode - mit Beispielen aus dem Schulfach Informatik. In: Log in vol. 3 (1983), Nr. 2, S. 16 – 20.

[FS82] Frey, Karl; Schäfer, Ulrich: Die Projektmethode. Beltz, Basel, 1982.

[FST13] Fuch, Alexander; Stolze, Carl; Thomas, Oliver: Von der klassischen zur agilen Softwareentwicklung. In: Praxis der Wirtschaftsinformatik vol. 290 (2013), S. 17–26.

[Gö12] Göttel, Timo: Agiler Informatikunterricht : Soziale Aspekte der professionellen Softwareentwicklung im Schulunterricht erfolgreich erfahrbar machen. University of Hamburg, 2012.

[Gu14] Gudjons, Herbert: Handlungsorientiert lehren und lernen. Verlag Julius Klinkhardt, Bad Heilbrunn, 2014.

[HNR07] Hartmann, Werner; Näf, Michael; Reichert, Raimond: Informatikunterricht planen und durchführen. Springer, 2007.

[Hu05] Humbert, Ludger: Didaktik der Informatik. Teubner, 2005.

[MH10] Meerbaum-Salant, Orni; Hazzan, Orit: An Agile Constructionist Mentoring Methodology for Software Projects in the High School. In: ACM Transactions on Computing Education vol. 9 (2010), Nr. 4, S. 1–29.

[Re05] Reinmann, Gabi: Innovation ohne Forschung? Ein Prädoyer für den Design-Based Research-Ansatz in der Lehr-Lernforschung. In: Unterrichtswissenschaft vol. 33 (2005), Nr. 1, S. 52–69.

[RG12] Romeike, Ralf; Göttel, Timo: Agile Projects in High School Computing Education – Emphasizing a Learners' Perspective. In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE'12). ACM, Hamburg, 2012.

[Ru04] Rumpe, Bernhard: Agile Modellierung mit UML. Codegenerierung, Testfälle, Refactoring. Springer, 2004.

[SS11] Schubert, Sigrid; Schwill, Andreas: Didaktik der Informatik. Springer, 2011.

[SS05] Schneider, Daniel K; Synteta, Paraskevi: Conception and implementation of rich pedagogical scenarios through collaborative portal sites, 2005. http://tecfa.unige.ch/proj/seed/catalog/docs/schneider-icool-final.pdf, abgerufen am 10.04.2015.

[St10] Strunz-Maireder, Edith: Pedagogical Content Knowledge. In: wissenplus vol. 28 (2010), Nr. 5, S. 41–44.

[We05] Weigend, Michael: Extreme Programming im Klassenraum. In: S. Friedrich (ed.): INFOS 2005. Dresden: GI- Edition - Lecture Notes in Informatikcs [LNI], 2005.

"Now they just start working, and organize themselves" First Results of Introducing Agile Practices in Lessons

Petra Kastl
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany
petra.kastl@fau.de

Ralf Romeike
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany
ralf.romeike@fau.de

ABSTRACT
This paper demonstrates a design-based-research (DBR) framework that provides the basis for refining agile practices and artefacts for school projects. It takes into account the pedagogical content knowledge of teachers by combining theory with practice. Based on qualitative analysis of material of the initial iteration of the DBR process, the findings provide an empirical basis demonstrating difficulties of teachers and students with sequential models. In addition, the preliminary findings substantiate the assumption that agile practices reduce teachers' effort in design and use of project-based-learning modules.

Categories and Subject Descriptors
• **Social and professional topics~K-12 education** • *Software and its engineering~Agile software development*

Keywords
Agile practices, agile methods, design-based research, project-based-learning, secondary CS education

1. INTRODUCTION
Project-based-learning (PBL) in computer science (CS) is typically conducted as software projects and constitutes a unique scenario: Professional software development has a long tradition of conducting software projects and hence provides several scientifically derived methodologies for organizing these projects. In order to provide students with a real life understanding of software engineering, it is general practice to refer to the same methodologies in CS lessons. For historical reasons, curricula follow sequential models, such as the waterfall model. Teachers are encouraged to combine objectives from pedagogical (PBL-) methodologies, such as cooperative and self-organized learning, with the methodologies derived from software engineering [11]. According to both, teachers' experiences as well as existing literature, school software projects suffer from numerous problems related to the sequential project setup [6]. This is in accordance with the experiences in professional software development, where the disadvantages of a sequential approach, such as long planning phases at the beginning and little flexibility in the later phases, leads to quality issues, breaking of deadlines and low customer satisfaction. Since 2001, agile methods are therefore getting increasingly popular among software companies in order to address these issues.

WiPSCE '15, November 09 - 11, 2015, London, United Kingdom

ACM 978-1-4503-3753-3/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2818314.2818336

The interesting question is, if the proclaimed benefits of agile development methodologies (e.g. an emphasis on team motivation, flexibility, and communication processes) and the known positive experiences [7] can be transferred into computing education. In particular, will students and teachers be able to solve present problems by using an agile approach and how much improvement can be demonstrated with respect to the objectives of PBL? Indeed, characteristics of PBL [e.g. 5] show similarities to descriptions of agile projects. For example, the emphasis on self-organized and outcome oriented effort, a complex and typically unclear defined task, as well as in cooperative work and learning combined with a collective responsibility for the project are important in PBL as well as in agile development. Furthermore in PBL, students aim to acquire, apply and enhance a variety of subject-related, methodological and social competencies [1].

The positive effects of applying agile approaches in CS school projects are e.g. discussed by Göttel [3], who used communication-supporting agile practices to facilitate an interest in and attractive notion of CS, as well as by Meerbaum-Salant and Hazzan [6], who developed a mentoring methodology for software projects based on agile values that aimed to support teachers. With respect to problems that students experience with software projects, Romeike and Göttel [10] proposed an Agile Model for Projects in Computing Education (AMoPCE) for use in school software projects. The practices presented in AMoPCE were used as a first guide for teachers who conducted the lessons that are discussed in this report and are referred to as *agile framework* or *agile practices (and artefacts)*.

2. THE RESEARCH APPROACH
Educational research and practice have shown that sustainable teaching innovations can hardly be implemented on the basis of theoretical considerations only. There is the need of input from pilot teaching which can reveal problems that have not been foreseen. Untested theory-derived teaching models may lead to significant issues in practice. However, CS education (CSE) is dependent on continuous technical and pedagogical discussion of scientific innovations, due to the dynamic character of CS. Design-based research (DBR) [2] is a research format that may address this challange. It supports the design and implementation of innovative teaching models and is characterized by the motivation to improve teaching and learning. This will be achieved in an iterative process of theoretical design and practical implementation phases where researchers and teachers work closely together on an equal footing. With DBR, aspects of teaching interventions can be researched within different contexts in order to develop empirically derived design principles. Reed and Guzdial [9] also suggest a DBR approach for CSE in order to avoid polarities such as discussions about "the only path to success". Instead, DBR provides the ability to explore ways

that describe interventions in relation to conditions that "lead to significant improvement in learning outcomes."

In our research [4], we apply a DBR process in order to use the pedagogical content knowledge (PCK) of experienced CS teachers with the goal of developing a field-tested and flexible-to-use set of agile practices and artefacts for school software projects. With this approach, we can take ideas, interpretations, individual adaptations and observations into account in order to successively enhance the agile framework with each iteration of the research process. The DBR process consists of phases for input and networking as well as of phases of transition into application. In the latter, teachers are encouraged to individually adapt or extend the agile practices.

3. RESEARCH QUESTIONS

So far there is only anecdotal evidence reporting problems with the waterfall model – which motivated the development of AMoPCE. Therefore, in the first iteration, we aim to solve the questions, whether there is empirical evidence for those problems and if we can comprehensively define the details on the problems with the adapted waterfall model that is used in many schools:

(1) Which problems and difficulties do students and teachers encounter when they apply a sequential process?

Based on own unpublished experiences and teachers' reports, we assume that the interviews will add data about a variety of problems that may impede the objectives of PBL. The data may also answer the question of how well characteristics and objectives of PBL are achieved with sequential process models.

(2) (Students): How do adapted agile methods support the objectives of PBL (better)?

An agile framework, when adapted to the learning setting by the teacher, may provide students with a meaningful structure and a practicable set of activities and tools to support students. We expect that this support will in practice have a positive effect on students' activities and will accentuate the objectives of PBL.

(3) (Teachers): Does an agile approach facilitate the planning, mentoring and assessment of SD projects for teachers?

The provision of students with well-defined practices carries several advantages for teachers. They will have to choose, adapt and introduce appropriate practices but may experience a change in the teaching-role since students work mainly independently. We expect that with an agile framework teachers will overcome previous obstacles related to the application of the sequential process model and that they are able to adapt the agile framework depending on the class-specific context.

4. METHODOLOGY

We started the DBR process with interested teachers, participating in an initial workshop where they got introduced to agile methods and AMoPCE as a corresponding model for CSE. The workshop encouraged teachers to adapt an agile framework to their individual in-class situation and to include own ideas. In the initial DBR iteration, six teachers applied the agile process in one 8th, three 9th two 10th and three 11th grade classes with approximately 170 students participating. In semi-structured interviews information about previous experiences with SD projects in CSE and about their motivations to apply an agile process this time was collected. Furthermore, qualitative data in relation to individual projects such as individual adaptations to fit the methodology to their context and individual observations and experiences were collected. For the analysis of the resulting data, a structured content analysis approach according to Mayring [8] was applied, in order to filter aspects of the material, which are relevant to answer the research questions. In order to guarantee the inter-coder reliability, the material was cross-coded in a peer-coding process.

The category system was inductively developed with respect to the research questions and the theoretical considerations related to AMoPCE. In the following, we sketch the main categories (italic): *Self-organization* comprises characteristics related to the learners' organizing of the product development and the associated activities as well as the learning and problem solving processes. The enhancement of professional, social and management skills as well as active and passive transfer of knowledge are summarized in the category *learning and social learning*. *Social interactions* encompasses the various aspects of communication and cooperation. Statements regarding the teachers' role and tasks are bundled in the category *teachers' activities*. The categories *transparency* and *assessment* are limited to aspects regarding the teachers in this report. The first one comprises aspects of keeping track of the project and its status as well as the insights into individual performances and the development of students' competencies. The latter encompasses the assessment of the achievements and the teachers' feedback to the students' individual developments and performances. Finally, *problems with sequential design models* refer to miscellaneous statements related to the first research question.

5. RESULTS OF THE FIRST ITERATION

5.1 Which Problems and Difficulties do Students and Teachers Encounter in a Sequential Design Process?

Problems with sequential design models were a topic in all interviews. After analysis of the material the following core problems were identified:

(1) Methodical competences: Since in sequential processes students only work once in each phase (analysis, design, coding and test), most of them are not able to familiarize with the used methods and activities. In addition they have too little opportunity to reflect on mistakes and to gain confidence. The point of an initial planning and design of a complex system which comprises a lot of considerations and decisions, remains purely theoretical for most of them as they do not experience and reflect on the consequences of missing or wrong decisions. Most students have not gained the ability to recognize and identify structural deficits in the planning. One teacher reflects: *"In principle I had to moderate each phase of the waterfall model[...] it was every time new for the students".*

(2) Self-organization: There is mutual agreement among teachers that students did gain almost no expertise in self-directed learning. This is highly likely related to the students' insufficient time for reflection and consequently the inability to develop confidence to master complex topics.

(3) Responsibility: Project-responsibility was mostly delegated to the teachers. Mistakes in the planning and design phases of a project are likely to lead to incompletion of the entire project. Thus teachers in general tightly guided the early phases. This role of the teacher typically did not change for the remaining project phases as well: *"They always saw the teacher as the leader of the process. The teacher is the one who knows and who can do everything […]. When we started coding, after two minutes hands were up."*

(4) Motivation: Teachers describe the influence of the sequential project layout as highly problematic for the students' motivation, which was high in the beginning but decreased rapidly: *"Before a small program finally is running, half a year is over and students' enthusiasm by then has been gone."*

(5) Social skills: The development of the students' social capabilities and competencies is mentioned by some teachers in reference

to observations with agile projects. As one teacher describes it, he was now able to observe, foster and assess the development of social competencies that he did not see in the years before: *"[In a sequential project], the student wouldn't have developed such social skills. At the end of the day he would have been the same nerd as he was right in the beginning."*

(6) Ending: Another problem that was substantiated in interviews was the ending of projects. In sequential projects, delays, which were common, had a direct and severe impact on the final project phases of coding and testing. *"Well, I also felt that the waterfall model gives me the problem of 'sink or swim'. Because what are you supposed to do with half-finished projects?"*

The analysis of the interviews supports the assumption that the teachers have difficulties in achieving the objectives of PBL, especially with respect to students' self-organization and motivation, methodical and social competencies as well as in taking responsibility for the project. Consequently, the question needs to be pursued, whether an agile framework is more suitable for reaching the objectives of PBL and to avoid the outlined problems of teachers.

5.2 How do Adapted Agile Methods Support Objectives of PBL (Better)?

Three of the main categories (outlined under 4) provide answers with a focus on the students' learning process using agile practices and artefacts. The findings of the evaluation of the material corroborate the hypothesis that in contrast to sequential models, an agile approach supports the objectives of PBL, in particular self-organization, social learning, communication and cooperation. Analysis of the data from interviews showed that students conduct their projects predominantly in a **self-organized** way. Due to the concrete guidelines of the practices, the students mainly had to closely follow the defined planning and communication processes. For example, with respect to problem solving and the acquisition of new skills a teacher explains how students handle programming problems: *"If they [the pair] don't succeed, they… pull the ripcord, saying 'hey, folks, we need a stand-up meeting [...]' And then they systematically approach the problem by making a list of all issues."*

Analysis of the product development process showed that students were able to self-organize e.g. in terms of students defining their sub-goals, splitting them into tasks, distributing and implementing the tasks, and communicating the results in the next stand-up meeting. Further observations were related to the self-organization of the phases (design, coding and test) which were part of each iteration: *"If students showed uncertainty about their upfront planning, they went to the project board, got the missing information and continued the coding work."* - *"They documented and categorized errors and tested them again in each iteration. So they even used a kind of regression testing."*

Learning and social learning became a clearly visible process due to the iterative development, as can be seen in the analyzed material. The increasing methodological competences were expressed e.g. in the above described self-organization of the phases of an iteration. The gradual increase of social and professional competences was evident throughout the interviews. As intended in PBL, the students acquired professional knowledge predominantly via self-directed learning and used it to achieve certain goals or to solve problems. All teachers valued pair-programming as a valuable method to support the passive transfer of knowledge. However, they also in unison pointed out that the regularly changing of the roles (driver/navigator) and the following of the roles requires the supervision of the teacher. Data from interviews suggest that over time arrangements and discussions are happening on a more regular basis and become more efficient. Spontaneously students took over roles as moderators or customers; sometimes the students chose fixed roles within the team. *"By now – and it has been quite different in the beginning – they have a project leader and this job is alternating. […] The timing of the stand-up meetings they now master quite well. Here they plan goal-oriented and split up after five to ten minutes […] to continue working on the computer."*

As one teacher explains it, special moments occurred when students cheered loudly after reaching a targeted goal: *"[…] especially positive is, when they get sense of achievement if they figure out something by themselves. I have never experienced before that students yell 'YES! YEEES!' and everyone gets together to have a look and say 'Wow, good job!'"*

Social interaction. The teachers considered communication important in all projects and hence extended the function of stand-up meetings (see below). This provided students with a convenient structure for successful collaboration and facilitated collective responsibility and intermediate success. At the beginning of the lessons, stand-up meetings were used by teams for information and coordination purposes. In consequence, transparency was increasing, an effect, on which a teacher reflects as follows: *"Everybody knew what the others were working on, which problems they encountered and which approaches they used, and which ideas the team came up with."*

Some teachers adapted stand-up meetings so that they could use them to spontaneously gather the students to discuss problems, for iteration planning or for the discussion of goals. In some cases, stand-up meetings have been used in order to conduct "talks with the customer", a possibility for the teacher to intervene in a steering or encouraging way or to reflect on process and outcome of an iteration. These adaptations of the professional methods demonstrate how teaching can benefit from practices intended to organize professional work flow.

Teachers also reported on loud and intense discussions. However, in all reported cases the conflicts were resolved by the students themselves. Without being able to explicitly explain how conflicts were resolved, a teacher summarized: *"It never happened that a student went off alone and started frustrated working by himself. Instead, students usually finished their meeting and at the end everyone went to the computer with well-defined tasks."* Similarly, another teacher emphasized the value of students' sharing common goals and responsibility: *"Ambition was awakened among most of the students. Only very few were holding back. They were willing to do it as a team and hence were taking care of their outcome."*

Most of the projects have been conducted early in the school year with novice programming students. Despite marginal previous knowledge, students were consistently able to manage their projects by themselves. Those teachers, who had the opportunity to compare the achievements with the outcomes of the previous year, considered them of higher-value.

5.3 Does an Agile Approach Facilitate the Planning, Mentoring and Assessment for Teachers?

A change in the effort of students should have a direct impact on the activities and role of teachers. In the following, we will discuss the outcomes of analyzing those main categories (outlined under 4), which refer to teachers.

The changes in **teachers' activities** were valued by all participating teachers and were comparable with the statements referring to the students. The data clearly show that the agile approach is sufficient to motivate the conduction of the project by students. Instead of a continuous moderation, only sporadic mentoring and advice was

used by teachers. The provision of support was described as much more relaxed, more intensive and more efficient. *"So I explained the principle once and then they worked […] Now I even had the opportunity to discuss issues in depth with the individual teams."- „Now there was time to help students even with simple errors like transposed letters, even if it took five minutes. I had the time, because I knew that the other students were working."*

Especially in the beginning, the teachers motivated and facilitated the communication and cooperation efforts of the students: *"In the beginning I pushed the students a bit: 'Wouldn't this be an opportunity to meet at the project-board to discuss?' And then over time I was surprised, they met regularly whenever they encountered a problem."* The flexibility of the requirements and the running prototypes after each iteration made planning and assessment easier. *"The students knew from the start that it's not about creating a perfect game but more about developing basic functionalities. This did not only change my own perception, but I had the feeling that also the students were positive about saying 'We have developed a prototype of game and we know how we could develop it further.'"*

The agile projects provided more **transparency** for the teachers in comparison to previous projects. On one hand, teachers gained insights into the progress of the projects and into students' activities by looking at the project board. This was beneficial for the provision of efficient support. *"You can actually choose which problem you want to address and at what time. I can't see anything from someone putting his hand up, but the project board tells me a lot."* On the other hand, due to the high significance of communication processes, teachers gained comprehensive insight into the individual progress of students: *"I found it totally interesting. You are not involved, you don't have to moderate – nothing. You can just listen. Of course, you notice much more; how they work, think, and how the process develops."-"You can better observe social behavior, and the group dynamics […] due to their constant interaction with each other." - "Indeed, by listening to them arguing about how they resolve the problems after mutual testing, you can determine that they are able to figure out if the problem lies in the coding or if they discuss 'we have to change something in the design of the class.'"*

Consequently, **assessment** is easier for teachers since they got a better understanding of the development process and can refer to individual prototypes. However, all teachers state that it was more important to them that, because of the increased transparency, they were able to give well-informed and comprehensive feedback with respect to individual performance, as it is expected in the PBL process. *"If I look at the individual progress of students, I have to say I observed significant improvements."* The interview statements support the hypothesis that an agile framework enables teachers to overcome significant difficulties that were present in sequential-model driven projects. Teachers adapted this framework to their individual situation and added own ideas. Such ideas included "student stories" to encourage subject-specific learning or "customer conversations" in order to guide projects. In the interviews there was no indication that teachers encountered difficulties to adapt agile practices to their context or to accomplish intended goals.

6. DISCUSSION

As one of the teachers stated, the effects of using agile practices in school software projects can be summarized as "Now they just start working, and organize themselves." The design is the core element of our DBR process. The iterative development of an agile framework for projects in CSE shall provide a framework that maintains a professional orientation and supports PBL at the same time. The results presented in this paper confirm the assumption that teachers can successfully create such a framework. Thus in a 2nd workshop teachers exchanged and reflected on their experiences. Together with the results presented above this provides the basis for further elaboration on agile practices being another step towards a set of agile practices as well as towards further contributions to theory.

Analysis of practical examples in the initial iteration showed that four out of six teachers applied the agile approach in classes early in the courses with programming novices. As they all valued the approach as profitable, we consider to reinvestigate the material with emphasis on students' self-efficacy and collective efficacy. Furthermore, the flexibility of the DBR process might be used to investigate how an agile approach enables teachers to structure projects in a way that they function as (collective) efficacy builders.

We were surprised about the passion and personal involvement of teachers. They adapted the approach courageously and creatively to their context, spending a lot of time for preparation. Therefore, a format which enables researchers and practitioners, both as experts in their field, to work together on equal footing has the potential to increase the sustainability of advanced teacher training. We are also surprised about the little trouble related to the application of the agile framework, which even was new to all of the teachers. We assume that the following aspects contributed to this success: Firstly we see the teachers' motivation and personal engagement, having their roots in the promising outlook to overcome real problems. As another success factor we see the combination of theoretical considerations as well as practical experience and PCK.

7. REFERENCES

[1] P. Blumfeld, E. Solowy, R. Marx, J. Krajcik, M. Guzdial, and A. Palincsar. Motivating Project-Based Learning: Sustaining the Doing Supporting the Learning. *Educational Psychologist* 26, 3 & 4, 369–398, 1991.

[2] Design-based Research Collective. Design-Based Research: An Emerging Paradigm for Educational Inquiry. *Educational Researcher* 32, 1, 5–8, 2003.

[3] T. Göttel. *Agiler Informatikunterricht: Soziale Aspekte der professionellen Softwareentwicklung im Schulunterricht erfolgreich erfahrbar machen*. Hamburg, 2012.

[4] P. Kastl and R. Romeike. Entwicklung eines agilen Frameworks mit Design Based Research. In *INFOS 2015 - 16. GI Fachtagung Informatik und Schule*. 2015.

[5] T. Markham, J. Larmer, and J. L.Ravitz. *Project based learning handbook. A guide to standards-focused project based learning for middle and high school teachers*. Buck Institute for Education, Novato, Calif., 2003.

[6] O. Meerbaum-Salant and O. Hazzan. An Agile Constructionist Mentoring Methodology for Software Projects in the High School. *ACM TOCE* 9, 4, 1–29, 2010.

[7] B. Meyer. *Agile! The good, the hype and the ugly*. Springer, 2014.

[8] P. Mayring. Qualitative Content Analysis. *Forum: Qualitative Social Research* 1, 2, 2000.

[9] D. Reed and M. Guzdial. The power of computing; design guidelines in CS education. *Commun. ACM* 55, 4, 8, 2012.

[10] R. Romeike and T. Göttel. Agile Projects in High School Computing Education: Emphasizing a Learners' Perspective. WiPSCE '12. ACM, NY, USA, 48–57, 2012.

[11] S. Schubert and A. Schwill. *Didaktik der Informatik*. Springer, 2011.