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.

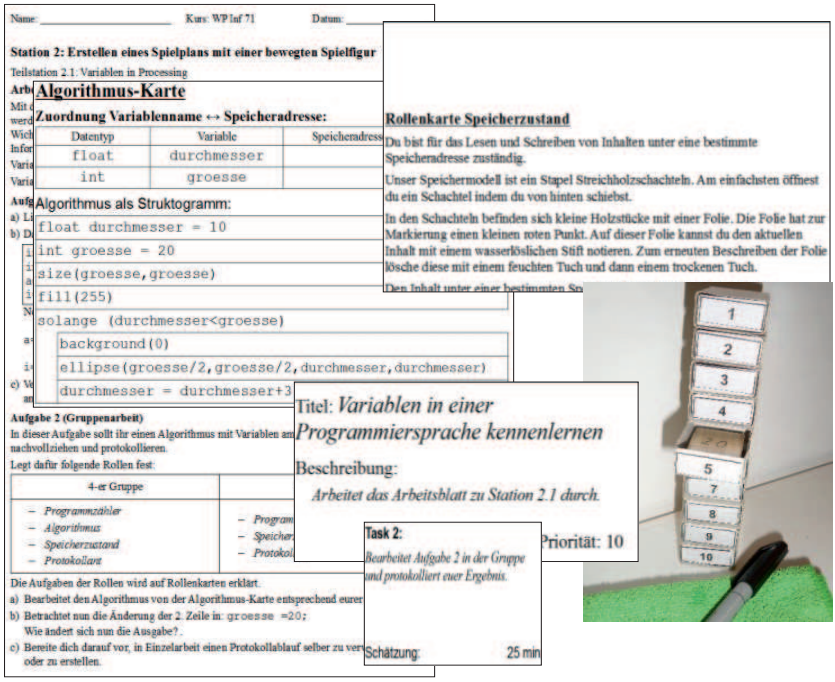

Abb. 2: Student Story zum Thema Variablen

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.