# Die Attraktivität der Informatik und soziale Aspekte der modernen Softwareentwicklung im Schulkontext

Dissertation zur Erlangung des Doktorgrades
an der Fakultät für Mathematik, Informatik und Naturwissenschaften,
Fachbereich Informatik
der Universität Hamburg

vorgelegt von

Diplom-Mediensystemwissenschaftler
Timo Göttel

Tag der mündlichen Prüfung: 27. März 2012

Gutachter:
Prof. Dr. Horst Oberquelle
Prof. Dr. Norbert Breier

Für Ilse, Kalle,
Katrin und Emil.

# Abstract

Computer science (CS) understands itself as being attractive and centered on social interaction. In contrast, society in particular adolescents recognize CS to be a soloist discipline. Hence, they hold a negative image of CS. Yet, focusing on social aspects in CS education can help to improve the image of CS.

In middle school, CS education is mainly organized in group work rarely influenced by professional software development. However, aspects of software development (namely agile methods) would allow highlighting social processes in educational contexts. These approaches can only be found in higher education. In addition, the research community on child computer interaction shows little interest in social aspects. Thus, adolescents cannot be reached to cultivate a positive image of CS. Consequently, this retains future talent shortage in the discipline.

Eight projects were conducted together with a total of 75 pupils (age 10–17). In these projects the tools *scratch* and *Greenfoot* were applied supported by agile methods. Working with the groups revealed several communicating issues and helped to compile requirements for computer supported learning. These can be summarized in three categories: Enable exchange, foster *awareness*, and support external presentation of outcomes.

To address these requirements, this thesis proposes a prototype which extends *scratch* with three modules: a group platform, a group chat, and a wiki. Hence, users can follow group activities and interact with each other. To provide applicability of the prototype in class, a list of essential adaptions to agile methods is suggested.

A focus on social aspects of the presented work is proved by meeting all requirements. Furthermore, *cognitive walkthroughs* and a weighted matrix from the perspectives of personas show positive results: Pupils who are interested in CS but who feel discouraged by falsely presumed underlying practices would strongly benefit from using the presented solutions. In conclusion, applying the proposed emphasis on social aspects to both tools and organization in class would help CS to become more attractive among adolescents.

**Keywords:** ***image of computer science, initial learning environments, agile methods, education***

# Zusammenfassung

Die Informatik begreift sich als attraktive Disziplin, die auf soziale Interaktion Wert legt. In der Gesellschaft und besonders bei Jugendlichen wird sie dagegen eher als eine Disziplin für Solisten wahrgenommen. Zur Widerlegung dieses Images empfiehlt sich die Fokussierung auf soziale Aspekte in der Informatikbildung.

Um Informatik im Schulkontext zu vermitteln, wird vorwiegend auf Gruppenarbeit gesetzt, die kaum Bezüge zur Softwareentwicklung aufweist. Ansätze aus der Softwareentwicklung – den agilen Methoden – fördern soziale Interaktion, werden jedoch nicht in der schulischen, sondern erst in der universitären Lehre eingesetzt. Auch die *Child Computer Interaction* berücksichtigt kaum soziale Interaktion. Dadurch wird verhindert, das Image der Informatik bei Jugendlichen nachhaltig zu beeinflussen. Der Disziplin gehen so kompetente Fachkräfte verloren.

Beobachtungen zu acht eigenen Projekten mit 75 Jugendlichen (10–17 Jahre) verdeutlichen, wie diese mit *scratch* bzw. *Greenfoot* und agilen Methoden arbeiten: Besondere Probleme existieren bei der Kommunikation und beim Austausch von Dateien. Daraus resultieren Anforderungen an die zukünftige Arbeit mit computergestützten Lernumgebungen, die sich in drei Kategorien einteilen lassen: Austauschmöglichkeiten anbieten, *awareness* schaffen und Außendarstellung fördern.

Um den Anforderungen gerecht zu werden, stellt diese Arbeit einen Prototyp vor, der *scratch* um eine Gruppenplattform, einen Gruppenchat und ein Wiki erweitert. Nutzer können damit die Aktivitäten der Gruppe verfolgen und kommunizieren. Die Ergänzung des Prototyps um die empfohlenen Anpassungen von agilen Methoden ermöglicht den Einsatz im Schulkontext.

Die Fokussierung auf soziale Aspekte der vorgestellten Maßnahmen wird durch die Abdeckung aller Anforderungen gewährleistet. *Cognitive walkthroughs* und eine Gewichtungsmatrix zu vier Personas verdeutlichen: Jugendliche, die Interesse an Informatik zeigen, jedoch von den dort vermeintlich vorherrschenden Vorgehensweisen abgeschreckt sind, profitieren von den vorgeschlagenen Maßnahmen. Die Einbeziehung sozialer Aspekte im Informatikunterricht trägt so dazu bei, dass die Disziplin von verschiedenen jugendlichen Zielgruppen als attraktiv wahrgenommen wird.

**Schlagwörter:** ***Image Informatik, Programmierlernumgebungen, agile Methoden, Schulkontext***

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1

# Einführung

## 1.1 Problemfeld

Sowohl die Wissenschaft als auch die Industrie und die Medien sind sich einig: auf absehbare Zeit wird es nicht genügend Studienabsolventen[1] der Informatik geben, um den immensen momentanen und zukünftigen Bedarf an Fachkräften der Informationstechnologie oder auch Informationstechnik (IT) zu decken. Sucht man nach Ursachen für die zu wenigen Studienanfänger und damit auch Studienabsolventen, stellt sich zwangsläufig die Frage, welches Ansehen die Informatik in der Gesellschaft inne hat, und ob dieses Bild mit der Wirklichkeit übereinstimmt. Geht man davon aus, dass dieses Image eher negativ geprägt ist, so gilt es herauszufinden, worin dies begründet ist. Auf Grundlage dieser Erkenntnisse lassen sich womöglich Werkzeuge und Methoden entwickeln, die in der Lage sind, möglichst frühzeitig ein zutreffenderes Image der Informatik zu fördern. Konkret erfordert dies, die gesellschaftlich selten wahrgenommenen sozialen Aspekte der modernen Softwareentwicklung für Jugendliche erfahrbar zu machen und eindeutig in Bezug zum realen Berufsbild zu setzen. Naheliegend ist es dabei, dort anzusetzen, wo Jugendliche das erste Mal strukturiert mit Informationstechnologie in Berührung kommen sollten: im Schulkontext, genauer gesagt, dem Anfangsunterricht. Sollte es dort gelingen, der IT zu einem dynamischen und sozialen Bild zu verhelfen, bestünde eine berechtigte Hoffnung auf eine Attraktivitätssteigerung der IT und somit im weiteren Bildungsverlauf auf einen Anstieg der Studienanfängerzahlen.

[1] Zur besseren Lesbarkeit der Arbeit schließt die darin durchgängig verwendete männliche Form immer auch die weibliche ein.

## 1.2 Aufgabenstellung

Ziel der Arbeit ist es, Werkzeuge und Methoden zu entwickeln und anzubieten, die soziale Aspekte der Softwareentwicklung in den Vordergrund stellen, um so die Attraktivität der Informatik zu verdeutlichen und adäquat bzw. realitätsnah darzustellen.

### 1.2.1 Überblick

Das erste Kapitel der vorliegenden Arbeit betrachtet die Diskrepanz zwischen dem Ansehen der Informatik und der realen Situation bei Methoden und Praktiken der modernen Softwareentwicklung mit eindeutiger Sozialorientierung. Ein Imageproblem wird daran deutlich, dass Informatik nicht nur in Deutschland weitläufig als langweilig, „*geeky*“ und intellektuell wenig anregend empfunden wird [Henriksen et al., 2010]. Henriksen et al. stellen ebenfalls fest, dass soziale Interaktion in keinster Weise mit dem Berufsbild Informatik assoziiert wird. Daher werden im mittleren Teil der vorliegenden Arbeit mögliche technische und methodisch-didaktische Lösungen vorgestellt, die deutlich soziale Aspekte der Informatik bereits in der schulischen Bildung hervorheben, um so ein realitätsnahes Bild des Berufsfelds zu vermitteln. In der vorliegenden Arbeit wird davon ausgegangen, dass auf diesem Weg eine hohe Anzahl von Jugendlichen von der Attraktivität der IT bzw. der Informatik überzeugt werden kann. Dies ist zwingend nötig, da auf absehbare Zeit der Bedarf an Informatikabsolventen wächst und bereits zum jetzigen Zeitpunkt zu niedrige Studienanfängerzahlen existieren. Dies gefährdet beispielsweise die Wirtschaftskraft eines Landes wie Deutschland, das exportorientiert ist und so abhängig von Innovation im Bereich der hochwertigen Technologien ist (vergleiche dazu [Anger et al., 2011]). Einen beobachteten positiven Aspekt sollte man auch gleich zu Anfang nennen: Es gibt zahlreiche verschiedene jugendliche Zielgruppen, die sich für die Informatik interessieren, jedoch im Laufe der Zeit (z.B. bis zur Wahl eines Studienfachs) das Interesse daran zu verlieren scheinen. Diese Personengruppen sind es, die mit den in dieser Arbeit vorgestellten Lösungsvorschlägen von der Attraktivität der Informatik überzeugt werden sollen.

In der vorliegenden Arbeit wird demnach die These aufgestellt, dass die sozialen Aspekte der modernen Softwareentwicklung und folglich der Informatik so früh wie möglich (idealerweise beim ersten schulischen Kontakt mit Informatikthemen) verdeutlicht und gelehrt werden müssen (vergleiche nochmals [Henriksen et al., 2010, S. 83]), um mehr Interessenten mit unterschiedlichen Fähigkeiten und Denkweisen

für die Informatik zu gewinnen. Die vorliegende Arbeit verdeutlicht mittels eines theoretischen Gerüsts mit etablierten Methoden des *Usability Engineering*, dass Änderungen in Entwicklungsumgebungen und an den Schulkontext angepasste Methoden der Softwareentwicklung sinnvoll sind, um die Sozialorientiertheit stärker in den Vordergrund zu stellen.

### 1.2.2 Einordnung der Arbeit

Entsprechend den Forderungen von Fothe und Friedrich [2011] nach bundesweitem regulärem Informatikunterricht und im Hinblick auf Studien, die Miniprojekte zur Steigerung der Aufmerksamkeit für Technikberufe in Frage stellen [MoMoTech, 2011], sei erwähnt, dass nicht davon auszugehen ist, dass die im Zuge dieser Arbeit durchgeführten Projekte bereits eine Steigerung der Attraktivität der Informatik nach sich ziehen. Vielmehr ist die Arbeit mit dem grundlegenden Verständnis geschrieben, dass die vorgestellten Praktiken nur nachhaltige positive Wirkungen auf das Image der Informatik erzielen können, wenn sie im regulären Informatikunterricht über größere Zeiträume eingesetzt werden.

Es existieren zahlreiche Publikationen (z.B. [Romeike, 2007b; Caspersen und Kölling, 2009]), die darauf hinweisen, dass gerade im Schulkontext das Verständnis der IT bzw. Informatik eng verbunden mit der Softwareentwicklung bzw. der Programmierung ist und als zentrales Element wahrgenommen wird. Daher bezieht sich diese Arbeit explizit auf die Softwareentwicklung, bzw. das Programmieren, und es wird davon ausgegangen, dass eine Verbesserung des Ansehens der Programmiertätigkeiten auch der Informatik im allgemeinen mit all ihren Facetten gerecht wird. Gleichwohl sei hier erwähnt, dass darüber hinaus weitere Aspekte und Inhalte die Informatik ausmachen und bildungsrelevant sind. Somit ist es wünschenswert beispielsweise Algorithmen und Datenstrukturen auf ihr Potenzial zur Verbesserung des Images der Informatik zu untersuchen.

## 1.3 Aufbau der Arbeit

Die wissenschaftliche Herangehensweise der Arbeit ist hauptsächlich von der Aktionsforschung und den Methoden der Softwaretechnik und Softwareergonomie beeinflusst. Beide Felder legen einen großen Wert auf partizipative Gestaltung: Die Aktionsforschung geht davon aus, dass Forschungsfragen in realen Umgebungen zu suchen und anzugehen sind. Die Softwaretechnik und Softwareergonomie fokussie-

ren die Einbeziehung der Nutzer und versuchen Systeme auf diesem Weg nutzerfreundlich zu gestalten. Diese Arbeit greift zu großen Teilen auf Erfahrungen aus Projekten mit Schülern zurück und stellt aus diesen Beobachtungen Vermutungen über die Gründe für ein einseitiges Bild der Informatik an. Daraufhin werden Lösungsansätze vorgestellt, die partizipativ mit Jugendlichen entwickelt wurden, um zukünftig jungen Menschen von Anfang an ein adäquates und damit attraktiveres Image der Informatik zu vermitteln. Folglich ist die Didaktik der Informatik eine weiteres Feld, das in dieser Arbeit berücksichtigt wird.

### 1.3.1 Methodik

Der methodische Aufbau der Arbeit ist wie folgt angelegt: Zunächst wird allgemein beleuchtet, welches Selbstverständnis in der Informatik existiert und welche gesellschaftliche Betrachtungsweisen in der Informatik vorherrschen bzw. auf diesem Selbstverständnis aufbauen; daraufhin wird untersucht, welche realen Eigenschaften der modernen, professionellen Softwareentwicklung großen Zuspruch finden, aber in der gesellschaftlichen Wahrnehmung unterrepräsentiert erscheinen.

Es gilt dann in einem weiteren Schritt Gründe für diese verzerrte Wahrnehmung zu identifizieren. Zwei Felder sind dabei besonders relevant, da sie einen strukturierten und beeinflussbaren Kontakt mit IT darstellen: Die Informatikbildung und das Forschungsfeld der *Child Computer Interaction*. Beide Gebiete werden auf ihren Anteil sozialer Aspekte untersucht.

Um eine weitere wichtige Komponente zum Verständnis des Ansehens der Informatik unter Jugendlichen abzudecken, ist es nötig, im Sinne der partizipativen Aktionsforschung, Projekte mit Schülern durchzuführen (vergleiche beispielsweise McNiff [2011]). Hierbei wurde bewusst entschieden, thematisch ähnliche, aber strukturell verschiedene Projekte mit Jugendlichen durchzuführen. So kann gewährleistet werden, dass Beobachtungen zu mehreren heterogenen Projekte mit verschiedenen Teilnehmern zu allgemeinen Erkenntnisse führen können (vergleiche Moraveji et al. [2007]).

Aus den theoretischen Untersuchungen und den praktischen Eindrücken werden dann Lösungen entwickelt, die es erlauben, soziale Aspekte der Informatik entsprechend der aktuell üblichen Methodologien der Softwareentwicklung in den Fokus zu rücken.

Konkret stellt diese Arbeit eine Erweiterung einer Programmierlernumgebung um *groupware*-Aspekte und methodisch-didaktische Anpassungen von agilen Methoden

für den Schulkontext vor. Es werden bewusst sowohl technische als auch methodische Ansätze verfolgt, um dem *blended learning* zu entsprechen (vergleiche dazu beispielsweise Osguthorpe und Graham [2003]).

Das vorgestellte Werkzeug und die Praktiken werden analytisch und mittels eines theoretischen Gerüsts auf ihre Attraktivitätssteigerung hin untersucht: Jugendliche Personas, die verschiedene Betrachtungsweisen erlauben und so einen möglichst objektiven Eindruck dieser Vorschläge vermitteln. Personas stellen verschiedene Nutzergruppen dar, die wiederum aus Betrachtungen der Projektarbeit abgeleitet werden konnten. Dieses analytische Evaluationsverfahren wurde verwendet, um einem Pygmalion-Effekt bei Evaluationen mit Schülern zu entgehen (vergleiche [Rosenthal und Jacobson, 1968; Rosenthal, 1995]). Diesem Effekt zufolge ist bei Jugendlichen, die zu einem vorgestellten Prototyp oder System zu dessen Attraktivität befragt werden, eine positive oder affirmative Haltung zu erwarten. Es ist anzunehmen, dass dieser Effekt auch in den durchgeführten Projekten vorhanden war. Jedoch kann man davon ausgehen, dass die dort zu erwartende Motivationssteigerung der Teilnehmer sich nicht negativ auf die computergestützte Gruppenarbeit auswirken. Vielmehr kann sogar davon ausgegangen werden, dass dadurch bedingte Probleme noch wohlwollend akzeptiert werden, wohingegen sie im alltäglichen Schulkontext sich als nicht akzeptabel erweisen würden.

### 1.3.2 Kapitelübersicht

Zunächst beleuchtet die Arbeit in Kapitel 2, was das Berufsfeld Informatik ausmacht und welches Ansehen die Disziplin unter Jugendlichen und in der Gesellschaft hat. So wird gezeigt, dass ein deutlicher Unterschied in gesellschaftlicher Wahrnehmung und Berufsrealität besteht: Zum einen ist das Image der Informatik von der Vorstellung geprägt, es handle sich um ein Feld, bei dem Solisten kryptische Anweisungen in den Computer tippen. Zum anderen existieren seit Jahren erfolgreiche und inzwischen weit verbreitete Vorgehensweisen in der Softwareentwicklung, im Besonderen die agilen Methoden. Diese legen verstärkt Wert auf kommunikative Prozesse und Teamarbeit. Darüber hinaus werden erste Indizien angeführt, die darauf schließen lassen, dass der Fokus auf soziale Interaktion in der Informatikbildung dieses Missverständnis aufheben könnte.

Kapitel 3 widmet sich der Frage, in welcher Weise die Informatik an Bildungsinstitutionen vermittelt wird und welche Rolle sozialorientierte Faktoren und Bezüge zur professionellen Softwareentwicklung dort spielen. Die Literatur zeigt deutlich,

dass der Einsatz und mögliche Anpassungen von agilen Methoden in der Hochschulbildung aktuell diskutiert werden. Darüber hinaus weisen vorherige Studien darauf hin, dass *pair programming* (eine Praktik der agilen Methoden) in der Informatik-Hochschullehre erfolgreich eingesetzt wird, sowohl um die Motivation und die Qualität der Ergebnisse zu steigern, als auch um soziale Aspekte des Programmierens vorteilhaft hervorzuheben. Eine Befragung unter Erstsemestern des Fachbereichs Informatik der Universität Hamburg wird in diesem Kapitel dargestellt und bestätigt die Ansicht aus der Literatur. Der positiv wahrgenommene Einsatz von *pair programming* in der Lehre erreicht momentan nur Personen, die sich für ein Informatik-Studium entschieden haben. Dieses Kapitel prüft gesondert, welches Image der Informatik an Schulen entstehen kann. Es werden Bildungsstandards in Bezug zu sozialen Komponenten hinterfragt und Literatur vorgestellt, die Gruppenarbeit oder Kollaboration mit Computerumgebungen thematisieren. Die Ergebnisse einer Umfrage unter Hamburger Informatiklehrern werden herangezogen, um die Annahme zu stützen, dass noch großer Handlungsbedarf besteht, um die Informatik sozialorientiert an Schulen zu präsentieren.

Die *Child Computer Interaction* (CCI) ist eine Disziplin, die den Umgang von Kindern und Jugendlichen mit Computern untersucht. In Kapitel 4 wird betrachtet, welche Erkenntnisse aus der CCI zu gewinnen sind und welche Systeme für jugendliche Novizen dort propagiert werden. In diesem Kontext werden zumeist technische Umgebungen vorgestellt, und so ist davon auszugehen, dass die dort entstandenen Systeme Einfluss auf das Ansehen von technischen Disziplinen bzw. der Informatik ausüben. Um dies zu prüfen, ist zu hinterfragen, welche Möglichkeiten zu Kooperation angeboten und hervorgehoben werden. Besonderes Augenmerk legt dieses Kapitel auf Kooperationsformen bei *digital storytelling* und *initial learning environments* (ILE), da beide Varianten Schüler dazu befähigen, (interaktive) Geschichten an Computern zu entwickeln.

Nach einem historischen Überblick über Programmiersprachen und Programmierumgebungen für Jugendliche und Kooperationsformen in Lernumgebungen werden die ILEs *scratch* und *Greenfoot* in Kaptiel 5 detailliert beschrieben und verglichen. ILEs stellen spezielle Programmierumgebungen für jugendliche Novizen dar, die entwickelt wurden, um die ersten Hürden des Programmiereinstiegs zu nehmen. Es wird bei aktuellen ILEs zumeist Wert darauf gelegt, jugendliche Interessen wie das Erstellen von Spielen oder von interaktiven Geschichten aufzugreifen und möglichst intuitiv zu ermöglichen.

Wie dieser Einsatz funktioniert, wird in Kapitel 6 näher erarbeitet. Dazu wurde im Rahmen dieser Dissertation die Verwendung der oben genannten ILEs in acht Projekten mit insgesamt 75 Schülern erprobt. Beobachtet wurden dabei jugendliche Kleingruppen (11-17 Jahre), die ein gemeinsames Ziel (interaktive Geschichte mit interkulturellem Kontext) in einer kurzen Projektzeit (1-5 Tage) erreichen sollten. Die Beobachtungen zeigen, wie Jugendliche in solchen Entwicklungsszenarien untereinander kommunizieren bzw. kooperieren und wie sie dabei von ILEs unterstützt werden.

Auf Grundlage der Beobachtungen zu den Projekten werden in Kapitel 7 entsprechende Anforderungen für Entwicklungsprozesse mit ILEs formuliert, um soziale Aspekte im Schulkontext positiv zu vermitteln. Diese Anforderungen leiten sich aus den Beobachtungen der Projekte mit Schülern und aus den vorgestellten Literaturquellen ab. Die genannten Anforderungen werden für eine bessere Übersicht in drei Kategorien unterteilt.

Unter Berücksichtigung der Anforderungen und in gemeinsamen Entwicklungsphasen mit Jugendlichen wurden Erweiterungen für das ILE *scratch* prototypisch unter dem Namen „der virtuelle Sandkasten“ umgesetzt und Praktiken der agilen Methoden für den Schulkontext angepasst. Der Prototyp und die methodisch-didaktischen Anpassungen der agilen Methoden werden in Kapitel 8 ausführlich vorgestellt.

Kapitel 9 vergleicht zunächst analytisch die Eigenschaften des virtuellen Sandkastens und der angepassten agilen Methoden mit den identifizierten Anforderungen. Dadurch wird belegt, dass diese durch die gewählten Maßnahmen abgedeckt sind und es ermöglichen, soziale Aspekte der Softwareentwicklung in den Vordergrund zu rücken. Darüber hinaus entstanden während der Projekte archetypische Schülergruppen bzw. jugendliche Personas, die es erlauben, ILEs oder auch Herangehensweisen auf ihren Einfluss auf das Ansehen der Informatik aus verschiedenen jugendlichen Perspektiven zu untersuchen. Nach einer allgemeinen Beschreibung des Persona-Konzepts in der modernen Softwareentwicklung werden die projektbasierten jugendlichen Personas präsentiert und verwendet, um das ursprüngliche *scratch* im herkömmlichen (Gruppen-) Unterrichtskontext mit der erweiterten prototypischen Version mit den agilen Methoden zu vergleichen und zu bewerten. In diesem Zuge kommen *cognitive walkthroughs* und eine Gewichtungsmatrix zum Einsatz.

Kapitel 10 fasst die Ergebnisse der Arbeit zusammen und diskutiert diese Erkenntnisse. Darüber hinaus werden mögliche Erweiterungen des virtuellen Sandkastens und des methodisch-didaktischen Vorgehens mit agilen Methoden präsentiert.

Es werden ebenfalls mögliche Szenarien zur Einbettung in den Schulalltag vorgestellt. Zusätzlich werden weitere Themenfelder umrissen, die ebenfalls einen positiven Beitrag zur Steigerung der Attraktivität der Informatik leisten können und demnach Gegenstand von weiteren Untersuchungen sein sollten.

# Kapitel 2

# Informatik und gesellschaftliche Wahrnehmung

Zu Beginn des folgenden Kapitels wird das Nachwuchsproblem der Informatik dargelegt. Im weiteren Verlauf werden Aspekte vorgestellt, die einen möglichen negativen Einfluss auf das Image der Informatik ausüben und somit eine Begründung für die zu niedrigen Studierendenzahlen darstellen.

Dazu wird das Selbstverständnis der Informatik aus wissenschaftlicher und beruflicher Sicht betrachtet. Dies zeigt, dass die IT tatsächlich eine kreative und gestaltende Disziplin ist. Am Beispiel der modernen Softwareentwicklung wird veranschaulicht, dass dieses Selbstbild zutrifft, es jedoch verpasst wird, dieses Bild Jugendlichen geeignet zu vermitteln.

Das Image der Informatik wird darauf hin mit der Sichtweise von Jugendlichen verglichen. Dies verdeutlicht, dass diese nicht mit dem Selbstbild der Informatik übereinstimmt und folglich Maßnahmen zu entwickeln sind, um diese Gegensätze aufzuheben.

Der Grund für diese Differenz wird weiterführend an einer unterrepräsentierten Nutzergruppe untersucht. Beispielhaft wird betrachtet, wie Frauen in der Softwareentwicklung gestärkt werden können. Es werden erste Indizien zusammengetragen, welche Aspekte auch männliche Personengruppen in der IT vermissen, so dass allgemein informatikinteressierte aber unentschlossene Personen von einer Studienwahl der Informatik zurückschrecken.

## 2.1 Informatik und Gesellschaft

Die IT ist der prägende Motor der heutigen Gesellschaft. Es gibt kaum noch Lebens- und Arbeitsbereiche, in die Computertechnik keinen Einzug gehalten hat, sowohl historisch visionär [Weiser, 1991] als auch aktuell [Denning, 2002; Borriello, 2006; Mattern und Floerkemeier, 2010] scheint das Verständnis vom allgegenwärtigen Computer weitestgehend gefestigt. Geht man davon aus, dass durch dieses Vorhandensein der Technik menschliche und soziale Abläufe modifiziert werden, so wird ersichtlich, welch erheblichen (positiven wie auch negativen) Einfluss die Programmierung von Software auf die Gesellschaft hat. Es wird hierbei nicht weiter auf Technikdeterminismus, Sozialkonstruktivismus[1], Technikoptimismus oder Technikpessimismus[2] im soziologischen Sinne eingegangen. Vielmehr wird davon ausgegangen, dass es sich um einen dialektischen Prozess von Technik und Gesellschaft handelt, bei der technische Entwicklungen immer in Wechselwirkung zur Gesellschaft stehen [Fuchs und Hofkirchner, 2003].

Gesellschaftlich scheint die Informatik nicht als attraktiv wahrgenommen zu werden, wie zu niedrige Studienanfängerzahlen, gemessen am Bedarf der Industrie und Aufnahmekapazitäten der Informatikstudiengänge, beweisen (ein Fallbeispiel für den Fachbereich Informatik der Universität Hamburg findet sich in Abbildung 2.1). Für 2010 wird zwar deutschlandweit von einem sehr leichten Anstieg der Studienanfängerzahlen berichtet. Ausgehend von der Gesamtzahl aller Erstsemester 2010 wird jedoch nach wie vor ein Defizit in der Informatik ausgemacht[3]. So verweist ein aktueller Trendbericht [Anger et al., 2011] auf eine strukturelle, nicht konjunkturell bedingte *MINT-Lücke*[4] im Allgemeinen für Deutschland. Im Bericht werden offene Stellen für Akademiker aus diesem Bereich mit der Anzahl arbeitsloser Akademiker über die Jahre 2000–2011 gegengerechnet [Anger et al., 2011, S. 17 f.]. Daraus ergibt sich auch im Falle der Informatik deutlich, dass der Bedarf an Akademikern zur Zeit nicht gedeckt ist. Auch in Zukunft ist, trotz leicht steigender Studienanfängerzahlen, nicht davon auszugehen. Der Report verweist im Gegenteil eher auf

[1] Beim Technikdeterminismus geht man davon aus, dass sämtliche gesellschaftliche Entwicklungen durch die Technik vorherbestimmt werden. Im Sozialkonstruktivismus geht man dagegen davon aus, dass nur durch gesellschaftliche Kontexte Technik entstehen kann [Degele, 2002, S. 28 ff., S. 98 ff.].

[2] Während der Technikoptimismus davon ausgeht, dass lediglich positive Effekte für die Gesellschaft durch Technik entstehen, geht der Technikpessimismus von dem Gegenteil aus [Fuchs und Hofkirchner, 2003, S. 236 f.].

[3] Siehe Mitteilung der Gesellschaft für Informatik e.V., http://goo.gl/6FvaF, zuletzt besucht am 15. April 2012.

[4] MINT steht für Mathematik, Informatik, Naturwissenschaften und Technik.

demographische Gegebenheiten hin, die darauf schließen lassen, dass der Anteil an Informatik-Absolventen in den nächsten Jahren sehr drastisch steigen müsste, um die hohe Zahl an bald pensionierten Fachkräften ausgleichen zu können. Beispielsweise müsste der Akademikeranteil der momentan 5–14-Jährigen ca. 30% betragen, was eine Verdopplung des Akademikeranteils zu der Kohorte der 45–54-Jährigen bedeuten würde [Anger et al., 2011, S. 37]. Darüber hinaus erinnert der Report daran, dass bei den aktuell, leicht positiven Absolventenzahlen zu berücksichtigen ist, dass die Neueinführung der Bachelor/Master-Studiengänge für kurze Zeit mehr Absolventen hervorbringt, da diese zeitgleich mit früher eingeschriebenen Diplomanden zum Abschluss führen [Anger et al., 2011, S. 42]. Außerdem sorgen die momentan noch vorhandenen doppelten Abiturjahrgänge für einen leichten Anstieg, der jedoch auf absehbare Zeit wieder verflachen wird [Anger et al., 2011, S. 42].

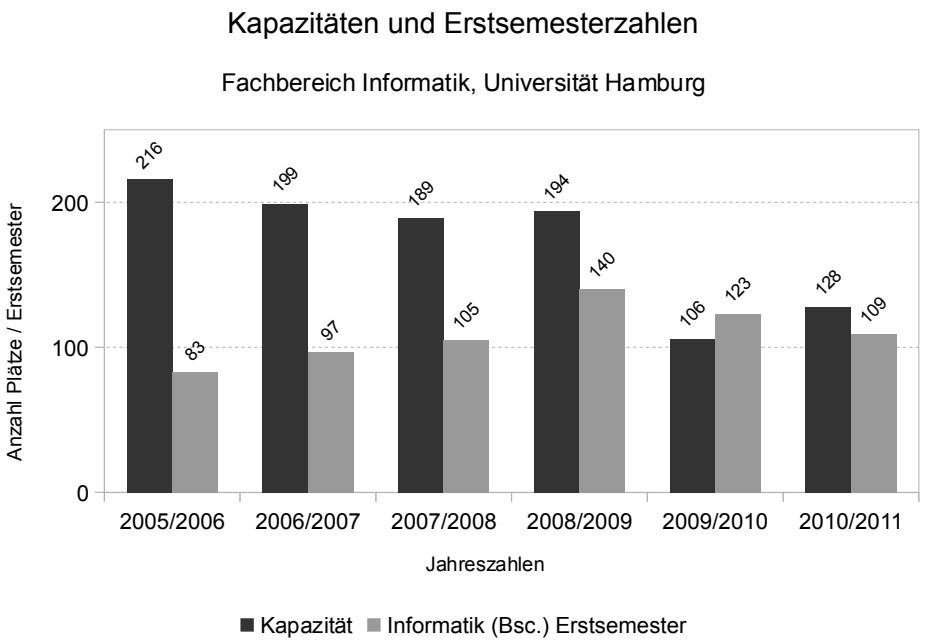



Abbildung 2.1: Aufnahmekapazität und Erstsemesterzahlen am Fachbereich Informatik der Universität Hamburg seit Einführung des Studiengangs Informatik *Bachelor of Science* (BSc). Spezialisierte BSc-Studiengänge ab Wintersemester 2009/2010 begründen Senkung der Kapazitätszahlen der Informatik. Quellen: [Universität Hamburg, 2005, 2006, 2007, 2008, 2009, 2010] und [Fachbereich Informatik der Universität Hamburg, 2011].

### 2.1.1 Selbstbild der Wissenschaft der Informatik

Zunächst wird eine Diskussionsgrundlage geschaffen, die die Nachwuchsproblematik im Spannungsfeld der Wissenschaft, Wirtschaft und der gesellschaftlichen Wahr-

nehmung der IT verortet. Dazu wird das Selbstverständnis von der Informatik als Wissenschaft zusammengefasst. Für den deutschsprachigen Raum gibt es dazu ein aussagekräftiges Positionspapier der Gesellschaft für Informatik (GI) [Gesellschaft für Informatik e.V., 2006], auf das an dieser Stelle eingegangen wird: Computer werden demnach unter anderem als Arbeitswerkzeug, Wissensspeicher, Regulator, Führungskraft und/oder (Unterhaltungs-) Medium wahrgenommen (a.a.O., S. 4). Die Informatik als zugrunde liegende Disziplin von Computertechnik soll Kern und Motor der Innovation sein (a.a.O., S. 5). In der Informatik ist die Information das zentrale Element, diese gilt es zu erstellen, abzubilden, zu speichern, anzuwenden und weiterzugeben (a.a.O., S. 6 f.). Durch Methoden aus der Informatik werden andere Wissenschaften und deren Informationsbearbeitungen beeinflusst und verändert, z.B. in der Medizin oder der Biologie (a.a.O., S. 19)[5].

Die GI beschreibt drei Facetten der Informatik: Zum einen ist sie Grundlagenwissenschaft ähnlich der Mathematik und zum anderen Ingenieurwissenschaft, die Lösungen für reale Probleme anderer Fachrichtungen implementiert. Als charakteristisch für die Ingenieurdisziplin wird hierbei die enge Zusammenarbeit mit Nutzern und Fachleuten aus anderen Fachgebieten angesehen. Zu Teilen wird die Informatik auch als Experimentalwissenschaft gesehen, die Szenarien simulieren und so virtuell erfahrbar machen kann. Alle drei Aspekte vereint nach Aussage der Autoren, dass Informatik ausgeprägt interdisziplinär angelegt ist (a.a.O., S. 8 ff.). Informatiksysteme sind heutzutage hochgradig in technische, wirtschaftliche und gesellschaftliche Systeme eingebettet und werden häufig gar nicht mehr als solche wahrgenommen. Der Informatik wird ein Verantwortungsbewusstsein attestiert, das sich auf Nutzen, kulturelle Auswirkungen und Nutzerorientiertheit bezieht (a.a.O., S. 11). Auch im Bildungsbereich hat die Informatik Änderungen hervorgerufen. Diese gilt es, durch angepasste Bildungsprozesse wie das *blended learning* aufzugreifen, welches E-Learning mit herkömmlichen Lehrmethoden verbindet und nur eine gemeinsame Verwendung zulässt (a.a.O., S. 26 f.). Auch auf kultureller Ebene ist längst evident, dass das Medium Computer eine Daseinsberechtigung als vereinendes Objekt von allen künstlerischen Ausdrucksarten besitzt. Es entwickelt sich immer mehr ein eigenes kulturelles Bild der Informatik, das dem einer Spielkultur mit verschiedenen Formen von sozialer Interaktion zu entsprechen scheint (a.a.O., S. 30 ff.). Die In-

[5]Ein weiterer Aspekt, der in diesem Zusammenhang nur indirekt wahrzunehmen ist, wird in eine anderen Quelle deutlicher genannt: Informatiker sind flexibel einsetzbar und nicht zwangsläufig reine Programmierer [Anger et al., 2011, S.3].

formatik verlangt nun, dass jeder Einzelne mit der immer größeren Informationsflut umzugehen lernt, um davon profitieren zu können.

Gesellschaftlich scheint die Informatik als zweischneidig wahrgenommen zu werden. Zum einen sind dort die offensichtlichen Vorteile der Informatik z.B. in Medizin, Technik oder Kommunikation. Zum anderen tragen einige für den Laien undurchsichtige Prozesse der IT zu einer Unsicherheit bei. Diese wird bestärkt durch den Einsatz von Informatiklösungen in der Arbeitswelt, die dadurch stark umstrukturiert wurde und wird (a.a.O., S. 34 f.). Zukünftig sieht die GI als wichtigsten Aspekt die Vertrauenswürdigkeit von Informatiksystemen an. Persönlichkeitsrechte sollen gewahrt werden, Systeme fehlerrobust, beherrschbar und benutzbar sein. Die Informatik ist aus Sicht der GI die dritte Säule der Entwicklung der Zukunft – neben den humanistischen und naturwissenschaftlichen Disziplinen (a.a.O., S. 36 f.). Bezüge zu sozialen Aspekten lassen sich in diesen Ausführungen nur implizit ableiten, so dass weitere Quellen daraufhin zu untersuchen sind.

Beispielsweise ist es Coy [2001] wichtig, dass sich die universitäre Informatik den gestaltenden und gestalterischen Aspekten in der Breite öffnet. Darüber hinaus ist er überzeugt, dass kommunikative und soziale Kompetenzen in der Informatik essentiell sind. Dieser Betrachtung nach hätte die Informatik zu begreifen, dass sie eher praktischen Disziplinen wie der Betriebswirtschaftslehre ähnelt, da nur ein umfassender, gut funktionierender Entwicklungsprozess zu guten Informatiklösungen führe. Er kommt dabei zu dem Schluss, dass viel zu lange aus wissenschaftspolitischen Blickwinkeln die Informatik gebildet wurde, ohne dabei das sich rasch entwickelnde Fach und die zugrunde liegende Technik zu beachten. So entstanden dem Autor nach Anwendungslücken, die häufig dazu führten, dass Informatikabsolventen von Universitäten keinen Bezug zu realen Prozessen hatten. Der Autor sieht jedoch gute Chancen, dass sich die meisten Universitäten dessen mittlerweile bewusst sind und dahingehend reagieren.

Auch in der einschlägigen Literatur zur Didaktik der Informatik finden sich die verschiedenen Aspekte dieses Selbstbilds der Informatik wieder. So versuchen sich beispielsweise Schubert und Schwill an einer Definition der Informatik und kommen zum Schluss, dass es für diese Disziplin und gerade für den Unterricht sinnvoll ist, mehrere Sichtweisen einzunehmen [Schubert und Schwill, 2011, Kapitel 1]. So sprechen sie von der Informatik als „Wissenschaft, die sich mit der systematischen und automatischen Verarbeitung, Speicherung und Übertragung von Daten aus Sicht der Hardware, der Software, der Grundlagen und der Auswirkungen befasst“ [Schubert und Schwill, 2011, S. 2]. Sie führen jedoch gleichzeitig an, dass es beispielsweise

auch eine „algorithmenorientierte“, eine „informationstheoretische“ und eine „arbeitsweltorientierte“ Sichtweise gibt, die jeweils über eigene Definitionen der Informatik verfügen und so zur Vielfältigkeit der Disziplin beitragen. Für Schubert und Schwill [2011, S. 5] ist es für den Unterricht entscheidend, dass man Informatik als eine Wissenschaft begreift, die den Entwurf und die Gestaltung von Informatiksystemen als zentrales Element beinhaltet. Dabei stellt ein Informatiksystem die Lösung eines Anwendungsproblems mittels Hardware, Software und Netzverbindungen dar. Auch Hubwieser lässt erkennen, dass es mehrere ähnliche Zugänge zur Informatik in der Schulbildung gibt (vgl. [Hubwieser, 2007, Teil B, Kapitel 1]). Zusammenfassend ist zu sagen, dass das Selbstbild der Informatik auch in der Didaktik viele Sichtweisen beinhaltet, die darauf abzielen, dass Systeme oder Algorithmen zu entwickeln sind ohne konkrete Bezüge zu sozialen Aspekten herauszustellen. Auch die Perspektiven der Schüler werden dabei vernachlässigt.

Zusammenfassend lässt sich festhalten, dass in der wissenschaftlichen Betrachtung entweder nur implizite Hinweise auf soziale Interaktion existieren oder sogar explizit ein Nachholbedarf in diesem Bereich gesehen wird. Ein Blick auf die berufliche Praxis hilft, die wissenschaftliche Sichtweise auf die Informatik zu ergänzen.

## 2.2 Sichtweisen der beruflichen IT: Soft skills und agile Methoden

Gemeinsam mit der Industrie wird versucht, aufgeschlüsselt nach verschiedenen Fähigkeiten oder Merkmalen, Anforderungen an Berufstätige in der IT-Branche zu definieren. Diese Anforderungen sind ein guter Indikator dafür, wie sich die Branche begreift und worauf sie Wert legt. Daher eignen sich diese Ansichten aus der Disziplin heraus auch gut für einen späteren Vergleich zu gesellschaftlichen Ansichten über die IT. Nachfolgend werden die wichtigsten Anforderungen vorgestellt, die aus wissenschaftlicher bzw. wirtschaftlicher Sicht definiert wurden.

Bei Betrachtung der entsprechenden Literatur der letzten zehn Jahre, wird ersichtlich, dass verschiedene soziale Fähigkeiten häufig von Universitätsabsolventen in der IT erwartet bzw. gewünscht werden. In einer Studie, die Führungspersonen aus zehn Unternehmen mit eigenen IT-Abteilungen zu den wichtigsten Fähigkeiten von Programmierern befragte, stellten beispielsweise Bailey und Stefaniak [2001] fest, dass unter insgesamt 23 Fähigkeiten, die von mehr als 75% der Befragten als besonders wichtig identifiziert wurden, sechs soziale Fähigkeiten aufgelistet waren.

Hierbei ging es z.B. um die Fähigkeiten zuhören, in der Gruppe arbeiten und mündlich kommunizieren zu können. Auch die Fähigkeit konstruktive Kritik zu äußern oder mit dieser umzugehen, wurde hoch angesehen. Reich und Nelson [2003] führten über fünfzig Interviews mit Leitern von IT-Abteilungen aus zweiundzwanzig verschiedenen Unternehmen. Aus diesen Interviews leiten sie für Beschäftigte der IT-Branche unter anderem die Notwendigkeit einer Stärkung von Team- und Kooperationsfähigkeiten in kulturell diversen und örtlich verteilten Kontexten ab. Darüber hinaus identifizieren sie eine erhöhte Nachfrage nach Führungsqualitäten unter IT-Fachpersonal. Reich und Nelson weisen ausdrücklich darauf hin, dass diese Fähigkeiten nur selten als Kompetenzziele in der Informatikbildung anzutreffen wären. Hall et al. [2007] konzentrieren sich in einer Studie vollkommen auf verschiedene Kommunikationsaspekte in der IT und versuchen mithilfe von Interviews und Fragebögen unter Beteiligten bei großen Softwareentwicklungsprojekten eine feinere Unterteilung zu erstellen. Demnach soll ein Programmierer, der von den anderen Gruppenmitgliedern als gut bezeichnet wird, vor allem proaktiv vorgehen, sein Wissen mit dem Team teilen und vollbrachte Arbeit vollständig und nachvollziehbar dokumentieren. Benamati [2007] führte dreizehn Interviews mit IT-Abteilungsleitern zu den gewünschten Fähigkeiten von Berufsanfängern. Auch hier erkennt Benamati, dass Kommunikationsfähigkeiten und Führungsqualitäten wertgeschätzt werden. Gleichzeitig wird in den Interviews häufig angemerkt, dass bei diesen Fähigkeiten nach wie vor Verbesserungspotenzial bei Universitätsabsolventen besteht. Auch neuere europäische Publikationen beweisen, dass diese Thematik aktuell und von hoher Bedeutung ist. So haben z.B. Kabicher et al. [2009] 128 IT-Stellenangebote von 85 Unternehmen im Umkreis von Wien auf ihre Anforderungen hin untersucht und mit den Ergebnissen aus Fragebögen von siebzehn Lehrenden und fünfunddreißig Arbeitgebern verglichen. Dabei stellten sie fest, dass in den Ausschreibungen neben Arbeitserfahrung Teamfähigkeit am häufigsten verlangt wird. Dies deckt sich auch mit den Ergebnissen ihrer Fragebögen, da Teamfähigkeit auch als ein Bereich von Lehrenden und Arbeitgebern angesehen wird, dem hohe Bedeutung eingeräumt wird und bei dem noch ein großes Verbesserungspotenzial bei Absolventen auszumachen ist. Im Vergleich stellen Kabicher et al. fest, dass vielfältige geistige Fähigkeiten, wie abstraktes, analytisches, zielorientiertes und innovatives Denken und das Verständnis für Zusammenhänge, äußerst gefragt sind.

Nachfolgend werden die getroffenen Aussagen zum abgeleiteten Selbstverständnis der IT-Branche noch einmal anhand der Praxis gefestigt: In der modernen Softwareentwicklung wird großer Wert auf die gemeinschaftliche Entwicklung in Teams und

auf soziale Aspekte gelegt. Hierzu eignen sich besonders die agilen Methoden, die eine aktuelle, viel gelobte Strömung in der Softwareentwicklung darstellen und in der Breite angewendet werden. Die agilen Methoden ermöglichen ein umfassendes Projektmanagement, mit dem die Kommunikation und die Abläufe zwischen Kunden und Entwicklern sowie innerhalb des Entwicklungsteams strukturiert und optimiert werden.

Agile Methoden folgen einem Manifest, das 2001 von Beck et al. [2001] verfasst wurde und bereits vorhandene populäre Herangehensweisen und Denkweisen der professionellen Softwareentwicklung bündeln sollte. Es entstand aus dem Wissen, dass große Projekte vormals immer darunter litten, dass Auftraggeber und Entwickler schnell unvereinbare Sichtweisen auf das Produkt entwickeln, wenn die Softwareentwicklung nach klassischen Modellen, wie z.B. dem Wasserfallmodell[6], aufgebaut ist. Ein leichtgewichtigeres Projektmanagement und klare Entwicklungsvorgaben, die dem Manifest folgen, waren demzufolge gefragt und sind bei allen Varianten der agilen Methoden immer als allumfassendes Gesamtpaket vorhanden. Das Manifest legt ein besonderes Augenmerk auf Werte, die hier im Original genannt sind:

> „Individuals and interactions over processes and tools
> Working software over comprehensive documentation
> Customer collaboration over contract negotiation
> Responding to change over following a plan“

Den Autoren zufolge sind Softwareentwicklungsprozesse dadurch mehr auf sozialen Austausch fokussiert und die entstehenden Produkte demnach besser an die Bedürfnisse der Nutzer angepasst. Hierbei ist zu erwähnen, dass es den Autoren bei den Entwicklungsvorgaben neben sozialeren Arbeitsstrukturen auch bzw. wohl in erster Linie um eine effektivere und befriedigendere Beziehung zwischen Auftraggeber und Dienstleister (Entwickler) geht: Ein inkrementelles und iteratives Vorgehen in kurzen Zyklen soll gewährleisten, dass Auftraggeber schneller lauffähige Ergebnisse erhalten und somit Spezifikationen eher angepasst werden können, denen die Entwickler wiederum folgen können.

[6]Das Wasserfallmodell z.B. nach Royce [1970] gehört zu den traditionellen Vorgehensweisen bzw. einem Projektmanagement der Softwareentwicklung. Es sieht sequentielle Arbeitsschritte vor, die ohne jegliche Rückkopplung abgearbeitet werden müssen, um zu einem fertigen Produkt zu gelangen. Diesem Modell wird vorgeworfen, dass die damit erzielten Softwareprodukte ungenügend sind, da das Vorgehen während der Entwicklung zu starr konzipiert ist und zu wenig kommunikative Prozesse darin beschrieben sind. Dadurch können häufig bestimmte Anforderungen der Nutzer nicht bedacht oder Änderungen im Anwendungsszenario nicht vorgenommen werden (vergleiche beispielsweise Larman und Basili [2003]). Dieses Vorgehen findet daher kaum noch Anklang in der aktuellen Softwareentwicklung.

Bis heute haben sich zwei Varianten der agilen Methoden besonders etabliert: Extreme Programming (XP) nach Beck [2000] und Scrum, das aus mehreren Arbeiten heraus entstanden ist, wobei drei Publikationen den größten Anteil daran zu haben scheinen [Takeuchi und Nonaka, 1986; Beedle et al., 1999; Schwaber und Beedle, 2001]. Beide Varianten versuchen, möglichst allumfassend den gesamten Prozess der Softwareentwicklung abzudecken und passende Praktiken anzubieten. So existieren z.B. auch zahlreiche Empfehlungen, wie komplexe Softwaresysteme strukturiert werden sollen und wie beim Programmieren vorzugehen ist. Im Sinne dieser Arbeit werden relevante Aspekte für den Schulkontext im Folgenden betrachtet [7], die auf soziale Strukturen bzw. Praktiken von XP und Scrum fokussiert sind.

Die Beschreibungen in den Abschnitten 2.2.1 bis 2.2.4 sind Göttel [2011a] entnommen und weichen nur in Teilen von dem dortigen Wortlaut ab.

### 2.2.1 Pair programming

Die bekannteste Praktik von XP dürfte das *pair programming* sein. Die Idee dabei ist, dass vor einem Rechner zwei Entwickler gemeinsam programmieren. In der Praxis bedeutet dies, dass eine Person die Tastatur verwendet und somit den eigentlichen Quelltext schreibt, wobei die zweite Person die Arbeit hinterfragt, auf Fehler prüft und alternative Lösungsvorschläge macht. Es ist besonders wichtig, dass die Person an der Tastatur die eigenen Gedanken bei der Arbeit mündlich mitteilt, um der zweiten Person das Erkennen von möglichen Denk- und Strukturfehlern zu erleichtern. Idealerweise beginnt die Person an der Tastatur erst zu schreiben, wenn sich das Paar über die Lösung einig ist. Im Laufe des Arbeitstages wird die Tastatur und damit die Rolle mehrmals untereinander getauscht.

### 2.2.2 User stories

Es handelt sich bei *user stories* um die kleinteilige und prägnante Beschreibung von elementaren Funktionalitäten, die einem Endnutzer mit dem fertigen Softwareprodukt zur Verfügung stehen sollen. Zu finden sind *user stories* sowohl bei XP als auch bei Scrum. Ziel ist es, diese elementaren Anforderungen nicht aus dem Auge zu verlieren, wobei die *user stories* gleichzeitig zur Organisation der Aufgaben innerhalb des Entwicklungsteams dienen. Häufig werden einzelne *user stories* dann während

[7] Es gibt vereinzelte Versuche, weitere Aspekte im Schulkontext anzuwenden, um die Qualität der dort erzielten Ergebnisse zu steigern (siehe [Weigend, 2005]). Darauf wird in Kapitel 3.2 noch einmal eingegangen.

*meetings* besprochen und den Programmierpaaren zur Fertigstellung zugeteilt. Erst nach erfolgreichem Abschluss einer *user story* widmet sich ein Paar neuen *user stories*. *User stories* dienen so der Kommunikation zwischen Kunde und Entwickler, aber auch zwischen den Entwicklern selbst.

### 2.2.3 Informative workspace

Bei XP wird empfohlen, dass sämtliche Prozesse und Lösungswege auf Postern oder ähnlichem skizziert werden. Diese Artefakte sollen in einem allgemein zugänglichen Raum zur Verfügung stehen, so dass sich Teilnehmer jederzeit über den aktuellen Stand der Entwicklung informieren können und darüber hinaus eine gemeinsame Diskussionsgrundlage haben.

### 2.2.4 Meetings

Eine Praktik, die man sowohl bei XP als auch Scrum findet, ist ein tägliches Treffen zu Beginn des Arbeitstages (XP: *stand up meeting*, Scrum: *daily scrum meeting*), bei dem alle Projektmitglieder kurz der Gruppe mitteilen, was am vergangenen Tag geschafft wurde, wo Probleme lagen und was am aktuellen Tag entwickelt werden soll.

### 2.2.5 Collective code ownership

Zu *collective code ownership* finden sich nur Literaturquellen für XP, es ist jedoch meist auch implizit in Scrum verankert. Es handelt sich hierbei um den Gedanken, dass das gesamte Entwicklungsteam für den vollständigen Quelltext verantwortlich ist und zu jeder Zeit Änderungen von jedem Teilnehmer der Gruppe im gesamten System erlaubt. So werden Abhängigkeiten von bestimmten Programmierern verhindert und eine Möglichkeit geboten, Module anderer Entwickler zu hinterfragen und diese zu diskutieren.

### 2.2.6 Sozialer Charakter der Praktiken

Diese fünf Praktiken der agilen Methoden stellen Teile des allgemeinen Projektmanagements dar und strukturieren so die Kommunikation und Kooperation innerhalb eines Entwicklungsteams. Das *pair programming*, der *informative workspace* und die *meetings* setzen direkt darauf, dass Informationen innerhalb der Teilnehmer weitergegeben und diskutiert werden. Die *user stories* und *collective code ownership*

stellen eher Praktiken dar, eine Infrastruktur zu schaffen, die jedem Teilnehmer eine Gruppenzugehörigkeit vermitteln und die Aufgabenverteilung innerhalb der Gruppe transparent darzulegen. Darauf aufbauend kann jeder Teilnehmer Aspekte ansprechen, die die Gruppe betreffen.

Auf Praktiken der agilen Methoden in der Informatikbildung wird in Kapitel 3.1.1 weiter eingegangen. In Kapitel 8.2 werden dann die oben genannten Praktiken noch einmal aufgegriffen, und es wird dargelegt, welche Probleme im Einsatz im Schulkontext auftreten können. Konkrete Anpassungen an diesen Kontext werden dort ebenfalls benannt.

Zusammenfassend lässt sich sagen, dass gerade soziale Aspekte von der Wirtschaft priorisiert werden und ein enormer Nachholbedarf in der Ausbildung erkannt wurde. Viele Universitäten versuchen zur Zeit darauf mit entsprechenden Anpassungen ihrer Curricula zu reagieren (vergleiche beispielsweise [Carter, 2011]). Eine andere Betrachtungsweise kann auch sein, dass dieses identifizierte Defizit in Teilen auch an der aktuellen Population von Studierenden in IT-Fächern festgemacht werden kann und somit bestimmte Personengruppen stärker zu fördern sind. So spricht Carter beispielsweise davon, dass der eigentlich unterrepräsentierten Gruppe der *Hacker* traditionell eine zu starke Aufmerksamkeit in der Informatikbildung zuteil wird (vergleiche Kapitel 9.3).

Festzuhalten bleibt, dass das Selbstverständnis der IT-Branche soziale Aspekte als relevant erachtet und immer wieder betont wird, dass kommunikative Gruppenprozesse eine Notwendigkeit in der IT sind. Wie bereits erwähnt ist dieser starke Fokus der Branche auf *soft skills* kaum bzw. nur implizit in den Aussagen der GI (siehe Seite 11) und den Lehrbüchern zur Didaktik der Informatik zu finden.

## 2.3 Gesellschaftlicher Blickwinkel

Die folgenden Quellen zeigen mögliche Gründe für die in Abschnitt 2.1 bereits erwähnten stagnierenden Studienanfängerzahlen in der Informatik auf.

Claus und Wilhelm sehen in der Informatik eine Wissenschaft der *unüberschaubaren Information*, bei der komplexe Programme von interdisziplinären Teams über mehrere Jahre entwickelt werden. Nach Claus und Wilhelm besteht Informatik trotzdem aus einem Zusammenspiel von Hard- und Software, die in allen Bereichen der Wirtschaft und Gesellschaft Relevanz finden und so die dortigen Prozesse verändern [Wilhelm, 1996, S. 9 ff.]. Spaniol fügt dem hinzu, dass die Informatik permanent mit Akzeptanzproblemen zu kämpfen hat. Dies führt er darauf zurück, dass bestimmte

Informatiklösungen damals wie heute eine gewisse Ungläubigkeit hervorrufen, die mit der Hinterfragung des Nutzens gepaart ist. Außerdem wird die Informatik laut Spaniol häufig als Wegbereiter für Rationalisierungen wahrgenommen. Der Autor erkennt die Angst der Gesellschaft, die Kontrolle zu verlieren, sich unmenschlicher Führung unterzuordnen oder Privatsphäre zu verlieren [Wilhelm, 1996, S. 133 f.].

Auch Denning [2002] ist wie die oben genannten Autoren überzeugt, dass die IT aus verschiedenen Gründen in der Gesellschaft einseitig wahrgenommen wird. Er stellt daher die grundsätzliche Frage, ob IT eine Profession darstellt. In diesem Zusammenhang bestätigt Denning das unvorteilhafte Ansehen der IT aus Sicht der Nutzer angesichts schlecht konzipierter Software, abstürzenden Systemen, komplexen und komplizierten Programmen, garantiefreier Software, unzureichendem technischen Support und Missachtung der Privatsphäre. Er sieht die IT von immer neuen Modewörtern (*buzz words*) geprägt, mit denen Nutzer meist wenig anfangen können und die sich obendrein dann auch noch entweder als banal oder unwichtig herausstellen. Oft stellen sie auch bloße Marketinginstrumente dar. Er ist sich sicher, dass IT-Lösungen noch daran kranken, dass die Entwickler häufig vergessen, dass die meisten Nutzer Pragmatiker sind, die nicht gewillt sind, Umwege oder suboptimale Lösungen hinzunehmen. Er führt dazu eine Aussage von Geoffrey Moore von 1991 an, der behauptet, es gäbe in der Gesellschaft fünf Typen, die jeweils unterschiedlich mit IT umgehen: Erfinder, Visionäre, Pragmatiker, Nachzügler/Zauderer und Ultrakonservative. Jede Gruppe ist in dieser Reihenfolge jeweils schwieriger vom Nutzen eines Systems zu überzeugen als die vorherige. Laut Moore ist von einer Glockenverteilung auszugehen, so dass in etwa 70-80 Prozent der Bevölkerung zu den Pragmatikern gehören. Genau den Übergang von den Visionären zu den Pragmatikern sehen sowohl Denning als auch Moore als problematisch an. Für Moore ist dies der Punkt, an dem IT-Unternehmen scheitern. Denning [2002] dagegen erkennt darin den Ursprung des schlechten Ansehens der IT: Die Entwickler verstehen die Beschwerden der Pragmatiker einfach nicht, besonders weil es doch bereits andere Gruppen gibt (Erfinder und Visionäre), die ihre Entwicklungen begeistert annehmen. Denning sieht hier das IT-Berufsfeld in der Pflicht, diese gesellschaftlichen Umstände wahrzunehmen und zu handeln.

Mit einem ähnlichen Tenor spricht Arno Rolf von den enormen Auswirkungen der IT in seinem Mikropolis-Modell. Er beschreibt in seinem Buch die Wechselwirkungen zwischen Informationstechnik und den Organisations- und Gesellschaftssystemen [Rolf, 2008a]. Er erläutert die Problematik, dass stets die IT für eine soziotechnische Wechselwirkung nach dem Mikropolis-Modell zuständig ist bzw. dafür

verantwortlich gemacht wird. Dieser Aspekt des Mikropolis-Modells beschreibt die Übertragung von menschlichen Handlungen in die IT und deren Rekontextualisierung in das ursprüngliche Betrachtungsfeld. Meist geht damit einher, dass sich die Nutzer dann an dem nun formalen Gerüst ausrichten müssen und nicht umgekehrt [Rolf, 2008a, S. 96 f.]. Implizit stellt sich hier auch die Frage, ob die IT somit als eine Domäne wahrgenommen wird, die die Anforderungen der Nutzer nicht versteht. Besonders im Feld der *social media* sieht Rolf ein riesiges Potenzial, das einerseits zur Durchsetzung globaler basisdemokratischer Ideen verhelfen kann (Wissensgesellschaft), gleichzeitig jedoch auch zu einer noch schnelleren Ökonomisierung führen kann (Wissensökonomie) [Rolf, 2008a, S. 69 ff.]. Folglich begreift der Autor eine Wissensgesellschaft – angetrieben durch IT – als eine Chance zur Verbesserung der gesellschaftlichen Sicht auf die IT. Der Autor geht daher auch implizit davon aus, dass gerade Informatiker bzw. Personen, die sich in einer informatisierten Welt zurechtfinden, die Welt nachhaltig und positiv mitgestalten können [Rolf, 2008a, S. 24]. Er sieht somit die Informationstechnik als eine Disziplin, die durch diese Gestaltungschancen eine hohe Attraktivität ausstrahlen sollte, und wundert sich daher über niedrige Studienanfängerzahlen an europäischen Universitäten [Rolf, 2008b].

Zusammenfassend und nicht verfrüht lässt sich also festhalten, dass die Informatik als eine Disziplin wahrgenommen wird, die rationalisiert, sozialen Kontakt verhindert und immer auch schwer begreifbare Gefahren (z.B. Verlust von Datenschutz) birgt. Dem gegenüber steht das Selbstbild der IT als kreative zukunftsweisende Disziplin, in der gemeinsame Lösungsstrategien mehr denn je gefragt sind. Das negative, gesellschaftliche Ansehen der Informatik kann besonders gut an den niedrigen Studienanfängerzahlen festgemacht werden. Daran lässt sich vermutlich ein ebenfalls einseitiges Image der Informatik unter Jugendlichen ableiten. Dieser Frage widmet sich der folgende Abschnitt.

### 2.3.1 Ansehen der Informatik unter Jugendlichen

Um Gründe für die zu niedrige Studienanfängerzahlen zu finden, ist es zielführend zu betrachten, wie Jugendliche und Studierende grundsätzlich die IT sehen. Eine Vielzahl von Publikationen geht auf diese Problematik ein, wovon nur einige hier genannt werden. Der Informatik werden laut einer Studie von García-Crespo et al. [2009] ausschließlich Einzel-Arbeitsprozesse zugeschrieben, nicht jedoch gemeinschaftliche und interdisziplinäre Arbeitsumgebungen. Wie schon in der Einleitung erwähnt, weisen Henriksen et al. [2010] darauf hin, dass die IT von Jugendlichen als nicht attrak-

tiv wahrgenommen wird. Eine umfassende und vor allem für den deutschsprachigen Raum relevante Arbeit liegt von Maass und Wiesner [2006] vor. Im Rahmen der Arbeit von Maass und Wiesner wurden Befragungen unter Schülern (11.-13. Klassenstufe, 216 Teilnehmer, davon 90 weiblich) als auch unter Studienanfängern der Informatik (84 Teilnehmer, davon 12 weiblich) durchgeführt. Sie wurden nach ihrem Bild der Informatik, ihren Erwartungen und ihrem Vorwissen befragt. Mit diesen Untersuchungen konnte gezeigt werden, dass Schüler Computerkenntnisse und Programmierkenntnisse als entscheidende Voraussetzung für ein Informatikstudium erkennen. *Social skills* dagegen spielen den Schülern zufolge nur eine stark untergeordnete Rolle, besonders weibliche Teilnehmer schätzen die Rolle noch deutlich geringer ein. Dieses verzerrte Bild sollte mit vertiefenden qualitativen Gruppendiskussionen an der Universität Bremen herausgestellt werden. Maass und Wiesner berichten, dass in diesen Diskussionen mit 30 Studierenden im Hauptstudium der Informatik (davon 10 weiblich) bestätigt wurde, dass Programmiervorkenntnisse nur eine untergeordnete Rolle spielen, wohingegen Teamfähigkeit und soziale Aspekte einen wesentlichen und positiv wahrgenommenen Teil des (Haupt-)Studiums ausmachen. Maass und Wiesner merken jedoch kritisch an, dass nur die wenigsten der ursprünglich interessierten und befähigten Schüler bis in ein Hauptstudium der Informatik gelangen, da sie ihr einseitiges Bild der Informatik bereits in der Schule oder auch in den ersten Semestern des Studiums zu erkennen glauben. Laut Maass und Wiesner ist der Festigung dieses negativen Bilds mit geeigneten und vielfältigen Maßnahmen entgegenzutreten – sowohl im Schul- als auch im Universitätskontext.

Im Verlauf von Projekten, die der vorliegenden Arbeit vorangingen kam in Gesprächen mit Schülern ebenfalls häufig die Auffassung zu Tage, dass es sich bei der typischen Programmiertätigkeit um reine Einzelarbeit handele. Im Gegensatz dazu sticht z.B. das weite Feld der *social media* heraus. Hierzu entstand in Gesprächen und Beobachtungen der Eindruck, dass Jugendliche einen selbstverständlichen aber auch ausgiebigen Umgang mit Computerumgebungen wie z.B. sozialen Netzwerken pflegen. Aufgrund dieses Gegensatzes erscheint es zweckmäßig, Schüler davon zu überzeugen, dass bereits die Entwicklung solcher Anwendungen ein kommunikativer Prozess und damit attraktiv für die Jugendlichen ist.

### 2.3.2 Indizien zur Verbesserung des Ansehens der IT

Über den geringen Frauenanteil in der Informatik wundert sich insbesondere Arno Rolf in seinem bereits erwähnten Artikel [Rolf, 2008b]. In der Tat scheint es viel ver-

sprechend, das Image der Informatik unter dem Aspekt zu beleuchten, warum es so wenig weibliche Studienanfänger gibt und welche Anforderungen Frauen an Berufe in der IT-Branche stellen. Burnett et al. [2010] befürworten weitergehend Verallgemeinerungen auf männliche Personengruppen solcher Betrachtungen und stützen sich dabei auf eine Arbeit von Ljungblad und Holmquist [2007]. Dort wird vorgeschlagen, sich Praktiken und Herangehensweisen von Subpopulationen, die nicht einmal zwangsläufig der Zielgruppe angehören müssen, in artverwandten Bereichen anzuschauen, um möglichst innovativ und gleichzeitig nutzernah interaktive Systeme gestalten zu können. Die Verfasser behaupten, dass auf diese Weise auch die Allgemeinheit der Nutzer von den entsprechenden Maßnahmen profitiert, da durch dieses Vorgehen menschliche Bedürfnisse leichter erkannt und darauf eingegangen werden könne [Ljungblad und Holmquist, 2007]. Burnett et al. [2010] gehen demnach davon aus, dass gerade durch die Berücksichtigung der Bedürfnisse der unterrepräsentierten Gruppe der Frauen positive Auswirkungen auf die Allgemeinheit erzielt werden. Dem liegt der Gedanke zugrunde, dass es nicht das typische Frauen- und Männerbild gibt, sondern nur noch Ausprägungen, die in verschiedener Intensität bei Männern und Frauen vorhanden sind. So sprechen nach Burnett et al. [2010] z.B. bestimmte Merkmale, die in der Allgemeinheit besonders Frauen bevorzugen, auch kleinere Gruppen von Männern an. Unter diesem Gesichtspunkt scheint es also adäquat, auf die Wünsche der Frauen an die IT einzugehen – wohlwissend, dass es auch Stimmen gibt, die den geringen Frauenanteil in der Informatik eher der gesellschaftlichen Prägung und der Bildung zuschreiben und daher die Generierung von Lösungsvorschlägen nicht vorrangig in der Informatik verorten [Teague, 1996; Clayton et al., 2009]. In Kapitel 9.3 wird eine männliche jugendliche Persona vorgestellt, die Interessen aufweist, die vielleicht spontan eher weiblichen Personen zugeschrieben werden. Dementsprechend wird dort deutlich, dass männliche Personen von Anpassungen profitieren können, die sich grob an den Bedürfnissen von Frauen ausrichten.

Berenson et al. [2004] berichten beispielsweise von Informatik-Studentinnen, denen gerade die Zusammenarbeit in der Softwareentwicklung wichtig ist. Die Interviewten sollten mehrere Aufgaben in unterschiedlichen Zusammenarbeitsformen lösen und wurden danach gefragt, welche Form ihnen am ehesten zusage und welche Schlüsse sie daraus zögen. Dabei ließ sich feststellen, dass *pair programming* (siehe Abschnitt 2.2.1 und 3.1.1) oder auch ganz allgemein Kooperation beim Programmieren die Zufriedenheit mit dem Ergebnis und damit auch das Selbstbewusstsein stärkt. Die Teilnehmerinnen bekräftigten, dass es ihnen neben dem Austausch über E-Mail- und Telefon auch auf ein persönliches Treffen der Teammitglieder ankomme.

Hierbei wurden die Befürchtungen geäußert, dass es in bestimmten Situationen beim Programmieren nicht möglich sei, einem Gegenüber Fragen zu stellen. Alle Teilnehmerinnen behaupteten dank der sozialen Aspekte der Programmierung, wie z.B. beim *pair programming*, für sich eine positive berufliche Zukunft in der Softwareentwicklung zu erkennen. Natürlich handelt es sich hierbei um eine qualitative Studie, die eine Verallgemeinerung nicht zulässt. Es sind jedoch Tendenzen erkennbar, die zu einem späteren Zeitpunkt dieser Arbeit (siehe 3.1.1) ebenfalls auftauchen, nämlich in einer internen Studie, die an der Universität Hamburg mit Studienanfängern durchgeführt wurde.

Burnett et al. [2010] suchen innerhalb der Programmierumgebungen nach Gründen, die Frauen von der IT fern halten. Sie erforschen die Wahrnehmung und die Nutzung von Programmierwerkzeugen in Abhängigkeit vom Geschlecht im beruflichen Kontext. In mehreren Studien beobachteten sie dazu Frauen und Männer im Umgang mit Entwicklungsumgebungen in verschiedenen Abteilungen eines großen Softwareentwicklungshauses. Sie stellten fest, dass Frauen meist bevorzugen, weniger Anwendungen parallel in Gebrauch zu haben. Außerdem scheinen sie weder so ausgeprägt nach spielerischen oder anderen Lösungswegen zu suchen, noch sind sie besonders daran interessiert, neue Features zu entdecken. Daraus und anhand der Aussagen der Teilnehmerinnen ergibt sich laut Burnett et al., dass Frauen deutlich weniger selbstbewusst eine bestimmte Aufgabe meistern. Im Gegensatz zu den männlichen Teilnehmern sehen sie sich nicht als Technikbegeisterte an. Während Erfahrung als Unterscheidungsmerkmal nur selten signifikante Werte lieferte, sind fast alle Unterschiede bezüglich des Geschlechts signifikant. Diese Aussagen lassen die Vermutung zu, dass bereits die Wirkung von Programmierumgebungen und eine bedachte Verwendung dieser in Gruppen die Einstellung zum Programmieren positiv beeinflussen kann.

Frauen sind bezüglich ihrer Einstellungen zu IT eine ausführlich erforschte Personengruppe[8]. Die Fülle der Ergebnisse und die Eindeutigkeit der Forderungen nach einem stärkerem Fokus auf soziale Aspekte in der IT kann demzufolge als ein allgemeiner geschlechtsunabhängiger Wunsch angesehen werden.

Zusammenfassend sei an dieser Stelle also erwähnt, dass gerade nicht in der IT vermutete soziale Aspekte positive Emotionen bei weiblichen Nutzern hervorrufen. Hier sei noch einmal an Maass und Wiesner [2006] erinnert und auf Kapitel 3.1.1

[8]So gibt es viele weitere umfangreiche Publikationen in diesem Umfeld, wie z.B. Margolis und Fisher [2001] oder auch Floyd et al. [2002], auf die im Rahmen dieser Arbeit nicht eingegangen wird.

verwiesen, in dem auf die genannten Aspekte in einer Studie mit Informatikanfängern an der Universität Hamburg eingegangen wird. Demzufolge sind nach Berenson et al. [2004] und Burnett et al. [2010] Lücken und Potenziale in der Informatikbildung und der CCI bezüglich sozialer Aspekte in dort genutzten Entwicklungsumgebungen zu identifizieren. Diese Betrachtungen liefern erste aussichtsreiche Indizien für mögliche Verbesserungen des Ansehens der IT. Das ist Gegenstand der nicht nur auf weibliche Nutzer bezogenen Betrachtungen in den drei folgenden Kapiteln.

## 2.4 Kapitelzusammenfassung

Zunächst wurde gezeigt, dass sich die Informatik als eine vielfältige Disziplin begreift, die es an einigen Stellen verpasst, auf soziale Interaktion deutlich hinzuweisen. Auch in der Wissenschaft war lange zu befürchten, dass das dort vermittelte Informatikwissen sich nicht ausreichend an realen Berufsbildern orientiert. So entstand der Eindruck, dass IT-Fachkräfte häufig qualitativ hochwertig ausgebildet sind, aber gerade bei sozialen Aspekten Nachholbedarf aufweisen. Dies zeigten auch die Betrachtungen zum Ansehen der Informatik unter Jugendlichen und zu Bedürfnissen von Frauen in Bezug auf IT und die Anforderungen an Berufsanfänger der IT-Branche.

Folglich wurde dargelegt, dass zwischen der realen Arbeitswelt in IT-Berufen, z.B. den agilen Methoden, und den verschiedenen negativen Ausprägungen des gesellschaftlichen Ansehens der Disziplin eine große Divergenz besteht: IT-Berufsbilder, die verstärkt auf agile Methoden setzen, zeichnen sich entgegen der landläufigen Meinung durch hochgradig interaktive, kreative und soziale Prozesse aus und können somit bei Jugendlichen bzw. Studienanfängern als durchaus attraktiv angesehen werden. Es wurde auch dargelegt, dass gerade Frauen IT attraktiver wahrnehmen, wenn *pair programming* im Informatikstudium eingesetzt wird.

Die Diversität der in diesem Kapitel genannten Punkte zum Ansehen der IT verdeutlicht, dass es nicht möglich sein wird, einen einzigen großen Hebel anzusetzen, um der Situation Herr zu werden. Statt dessen kann davon ausgegangen werden, dass es einer Vielzahl von verschiedenen Schritten bedarf, um der Informatik zu einem zutreffenden Ansehen zu verhelfen.

In den nächsten beiden Kapiteln wird der allgemeine gesellschaftliche Bereich verlassen. Es findet eine Fokussierung auf die Informatikbildung und die CCI statt, um weitere Aussagen zum Ansehen der Informatik und der dort vermittelten zwischenmenschlichen Interaktion treffen zu können.

# Kapitel 3

# Informatikbildung und soziale Aspekte

Soziale Aspekte der Softwareentwicklung scheinen also gesellschaftlich nicht wahrgenommen zu werden oder diese haben kein ausreichendes Gewicht, um einen positiven Einfluss ausüben zu können. Erste Indizien sprechen für den Einsatz von *pair programming*, um eine höhere Aufmerksamkeit zu erhalten. In diesem Kapitel werden daher Aspekte der agilen Methoden vorgestellt, die bereits in der Literatur zur Informatikbildung zu finden sind. Dabei zeigt sich, dass agile Methoden positiv in der Lehre aufgenommen, aber zur momentanen Zeit noch nicht in voller Breite eingesetzt werden, so dass sie nicht als essentiell für die Informatik wahrgenommen werden. Die Erkenntnisse aus der Literatur werden anhand von Ergebnissen einer Umfrage am Fachbereich Informatik der Universität Hamburg validiert.

Im zweiten Teil des Kapitels wird die Informatikbildung an Schulen genauer beleuchtet, da agile Methoden dort nach Literaturlage selten bis gar nicht zum Einsatz kommen. Dazu werden unter anderem die Bildungsstandards der GI und deutschsprachige Veröffentlichungen zur Didaktik der Informatik, sowie die Ergebnisse einer Befragung unter Informatiklehrern aus Hamburg und Schleswig-Holstein vorgestellt. Diese Betrachtungen belegen, dass soziale Aspekte im Informatikunterricht eine untergeordnete Rolle spielen und dort kaum Bezüge zur professionellen Softwareentwicklung hergestellt werden.

## 3.1 Relevanz sozialer Aspekte in der Bildung

In Kapitel 2.3.2 wurde in den Überlegungen zu Berenson et al. [2004] und Burnett et al. [2010] deutlich, dass im Bildungskontext soziale Aspekte in der Softwareentwicklung und der dort genutzten Programmierumgebungen einen positiven Einfluss auf das Ansehen der IT ausüben. Daher ist es sinnvoll, weitere Betrachtungen dazu anzustellen. Auch Fisker et al. [2008] weisen darauf hin, dass in der Ausbildung der Informatik bzw. der Softwareentwicklung so früh und so oft wie möglich Gruppenarbeit angeboten werden muss, da diese Arbeitsform dem Standard in der Industrie entspricht. Fisker et al. stellen drei Aspekte vor, die soziale Komponenten darstellen und von Entwicklungsumgebungen im Bildungskontext ihrer Meinung nach nativ unterstützt werden sollten: Austausch von Artefakten, Kommunikation und *awareness*. In ihrer Arbeit stellen Fisker et al. ein Konzept vor, das die Entwicklungsumgebung *BlueJ* [Kölling, 2008] um die Möglichkeit einer Versionsverwaltung erweitert und sich folglich auf den Austausch von Artefakten fokussiert. Dabei stellten sie fest, dass dafür auch Kommunikation und *awareness* mindestens rudimentär unterstützt werden müssen.

Der Begriff *awareness* kommt aus der computergestützten Kooperation (CGK) und umschreibt dort die Notwendigkeit einzelner Gruppenteilnehmer, sich der Gruppe und anderer Teilnehmer gewahr werden zu können. *Awareness* unterscheiden Gutwin et al. [1996] in vier verschiedene Kategorien: Informelle *awareness*, soziale *awareness*, *awareness* über Gruppe und Struktur und *awareness* über Arbeitsplatzsituationen. Die informelle *awareness* beschreibt das Gefühl dafür, wer momentan im Gruppenkontext arbeitet und welche Aufgaben die Teilnehmer zum momentanen Zeitpunkt ausführen. Die soziale *awareness* setzt einen Teilnehmer darüber in Kenntnis, welchen Gemütszustand (z.B. aufmerksam, gelangweilt, interessiert) die anderen Teilnehmer momentan haben. Bei der *awareness* über Gruppe und Struktur werden dem Teilnehmer Informationen über Rollen der anderen Teilnehmer bewusst. Die *awareness* über Arbeitsplatzsituationen bezieht sich auf das Bewusstsein über die vorhandene Infrastruktur am Arbeitsplatz und die Interaktion der einzelnen Teilnehmer mit dieser. Technisch dargebotene Informationen zur Steigerung der *awareness* werden insbesondere in Situationen bereitgestellt, wenn sich Gruppenteilnehmer nicht am selben Ort befinden können. Es hat sich herausgestellt, dass diese Zusatzinformationen für Gruppenteilnehmer wichtig sind, um erfolgreich arbeiten und sich innerhalb der Gruppe wohlfühlen zu können. Somit stellen sie auch Voraussetzungen für Lernumgebungen dar. Gleichwohl ist darauf zu achten, dass

*awareness*-Funktionen zurückhaltend gestaltet werden und den Nutzern keinen unverhältnismäßigen Zusatzaufwand abverlangen.

Während diese Quellen davon zeugen, dass soziale Aspekte bei computergestützten Lernumgebungen relevant sind, existieren weitere Betrachtungen, die sich an die agilen Methoden anlehnen und deren Vorteile beschreiben.

### 3.1.1 Agile Methoden in der Bildung

Betrachtet man Werte wie *Mut*, *Kommunikation*, *Feedback*, *Respekt* und *Einfachheit*, die die agilen Methoden ausmachen [Beck und Andres, 2004], so erscheint es klar, dass eine Überführung in einen Bildungskontext sinnvoll ist. Nachfolgend wird darauf eingegangen, bevor in Kapitel 8.2 notwendige Anpassungen für den Schulkontext vorgestellt werden, die im Rahmen dieser Arbeit entstanden sind.

In der Literatur finden sich zwei Themenschwerpunkte, in denen Praktiken der agilen Methoden zum Einsatz kommen: Einige Arbeiten versuchen, konkrete Programmierkonzepte oder die kompletten Vorgehensweisen der agilen Methoden einzusetzen, um die Einstiegshürden für Novizen zu minimieren, die Ergebnisse der Teilnehmer zu verbessern und Absolventen besser auf die reale Arbeitswelt der Softwareentwicklung vorzubereiten. Der Fokus liegt zumeist eher auf einem möglichst guten Lernergebnis als auf einer sozialen Umgebung mit Bezug zur realen professionellen Softwareentwicklung. Andere Arbeiten dagegen legen ihren Fokus auf eine einzelne Praktik der agilen Methoden, nämlich das *pair programming*, um Kommunikation und Kooperation bei Novizen zu fördern und so eine bessere Lernumgebung zu schaffen.

Immer mehr Autoren plädieren dafür, dass agile Methoden früh und umfassend in der Informatikbildung eine Rolle spielen sollten. Beispielhaft seien hier zehn Gründe genannt, die laut Hazzan und Dubinsky [2007] für den Einsatz der agilen Methoden an Hochschulkursen sprechen und bewusst plakativ benannt wurden, sich aber gut als Einleitung eignen:

> Agile Methoden...
>
> ... sind in der professionellen Softwareentwicklung weiterhin erfolgreich.
>
> ... schulen das Arbeiten im Team.
>
> ... beleuchten menschliche Aspekte.
>
> ... fördern Vielfältigkeit.
>
> ... unterstützen Lernprozesse.

... lehren Reflektions- und Abstraktionsvermögen.

... heben die Wichtigkeit eines Projektmanagements hervor.

... verbessern ethische Normen.

... zeichnen ein umfassendes Bild des Entwicklungsprozesses.

... stellen eine komplette Lern- und Lehrumgebung bereit.

### Umfassende Vorgehensmodelle

Eine Vielzahl von Arbeiten berichtet vom Einsatz von agilen Methoden als mehr oder weniger komplettes Vorgehensmodell [Caspersen und Kölling, 2009; Keefe et al., 2006; Loftus und Ratcliffe, 2005; McKinney und Denton, 2005; Rico und Sayani, 2009; Schild et al., 2010; Schneider und Johnston, 2005]. Den Arbeiten ist gemein, dass sie Vorteile in den agilen Methoden für die Lehre ausmachen, manche jedoch berichten von ungeahnten Hürden, um die Studierenden von der Sinnhaftigkeit des Vorgehens zu überzeugen. Einige Beobachtungen aus den Artikeln seien hier exemplarisch präsentiert, um das Verständnis für agile Methoden in der Bildung zu stärken.

Caspersen und Kölling [2009] weisen darauf hin, dass Programmieranfänger häufig ohne konkrete Arbeitsprozesse Programmierkonzepte vermittelt bekommen. Sie beschreiben herkömmliche Herangehensweisen in Programmier-Einführungsveranstaltungen an Universitäten als abschreckend für Novizen, da dort häufig komplexe Problemlösungen vorgestellt werden, als seien sie ad hoc von einer Person programmiert worden. Caspersen und Kölling monieren, dass Novizen auf diese Weise nicht in ausreichendem Maße erfahren, dass Programmieren immer einen Prozess darstellt, in dem strukturiert und kleinteilig Lösungen erarbeitet werden, und es darüber hinaus zu dem Prozess gehört, Fehler oder falsche Annahmen hinzunehmen und diese zu umgehen. Sie stellen die These auf, dass folglich viele Studierende in den ersten zwei bis drei Semestern das Informatik-Studium aufgeben, obwohl sie durchaus dafür geeignet wären. Um diesem Phänomen entgegenzuwirken, stellen sie ein Konzept vor, das Anfänger anleitet, Schritt für Schritt Klassen zu erstellen und mit Inhalten zu füllen. Diese Vorschläge sind zu großen Teilen aus Praktiken der agilen Methoden abgeleitet. Caspersen und Kölling erwähnen, dass die agilen Methoden in ihrer kompletten Breite nicht in Einführungsveranstaltungen angewendet werden können, da der Mehraufwand für Dozenten und Studierende zu groß sei – besonders unter dem Aspekt, dass solche Veranstaltungen aus ihrer Sicht häufig bereits überladen sind.

Loftus und Ratcliffe [2005] sind dagegen der Meinung, dass XP möglichst in seiner Ganzheit in der Lehre eingesetzt werden muss, damit Studierende sowohl die Stärken als auch die Schwächen praktisch erfahren. Konkret halten sie auch fest, dass losgelöste einzelne Methoden nicht ausreichen, da sie der Meinung sind, dass sich die einzelnen Methoden ergänzen. Damit relativieren sie die positiven Publikationen zum *pair programming* in der Lehre, die in Kapitel 3.1.1 vorgestellt werden. Loftus und Ratcliffe sind der Ansicht, dass XP neben dem Erlernen und Reflektieren der Methodologie selbst vier Lernerfolge aufweisen kann: effektives Erlernen neuer Technologien, Kommunikation, Planung, verantwortungsbewusstes Verhalten. Gerade die letzten drei Punkte stellen somit soziale Aspekte dar, die Einzug in die Informatikbildung erhalten.

Rico und Sayani [2009] berichten von einer nahezu kompletten Umsetzung der Vorgehensmodelle nach den agilen Methoden[1] in einem 13 wöchigen abschließenden Master-Kurs. Die Autoren kommen unter anderem zu dem Schluss, dass die entstandenen Softwareprodukte dank des Einsatzes der agilen Methoden eine hohe Qualität erreicht haben. Gleichzeitig berichten sie jedoch von Problemen, die Studierenden davon zu überzeugen, dass die Herangehensweise der agilen Methoden lohnenswert ist. Sie beobachteten in diesem Zusammenhang, dass sich viele Teilnehmer den Ideen widersetzten und es bevorzugten, nach bewährten Konzepten zu programmieren, die an klassische Methoden erinnern. Daher stellen Rico und Sayani auch heraus, dass die Grundideen der agilen Methoden deutlich früher in Einführungskursen vermittelt werden müssten, um noch erfolgreicher im Hochschulkontext eingesetzt zu werden. Sie merken auch an, dass die Gruppen, die gewillt waren, vollständig im Team zu arbeiten, deutlich besser abschnitten als Gruppen, bei denen trotz der agilen Methoden immer wieder Einzelarbeit auftrat.

Schild et al. [2010] berichten von einer erfolgreichen Anpassung und Umsetzung von *Scrum* in einem fortgeschrittenen Softwareentwicklungskurs (Game Development). Sie sind der Überzeugung, dass *Scrum* die Produktivität der Teilnehmer steigert, technisch aufwendige Lösungen als Ergebnisse liefert und damit auch trotz hohem Aufwand die Zufriedenheit der Teilnehmer steigert. Auch sie merken jedoch an, dass Studierende zuweilen versuchten, Praktiken von *Scrum* zu meiden bzw. nur die vorgegebene Mindestanforderung ablieferten (Produkt Backlog) und intern eigene Strukturen für die Entwicklung verwendeten. Auch hier kann man also davon

[1] Die Teilnehmer konnten frei wählen, welche Methode sie anwenden, die meisten entschieden sich für *Scrum*.

ausgehen, dass der Einsatz von agilen Methoden den vorher ausgebildeten Arbeitsstrukturen der Studierenden widerspricht und somit auf Widerstand trifft.

Auch Schneider und Johnston [2005] sehen agile Methoden als ein Gesamtpaket und hinterfragen, ob dieses an Hochschulen gelehrt werden kann. Sie sehen darin die Chance, die Studierenden auf sich ständig wechselnde Situationen vorzubereiten, und erhoffen sich die Förderung von kommunikativen Fähigkeiten. Schneider und Johnston sind der Ansicht, dass die immer wieder auftretenden Probleme der agilen Methoden in fortgeschrittenen Kursen zu großen Teilen darauf zurückzuführen sind, dass agile Methoden auf gemeinsame Lernprozesse und Lösungen setzen. Dies sehen die Autoren nämlich als Gegensatz zur Fokussierung der Studierenden auf gute Noten und Kurse, die im Laufe des Studiums mit herkömmlichen Lehrmethoden gefördert wird. Dementsprechend fordern auch sie eine frühe Verwendung von agilen Methoden in der Lehre und hoffen sogar, damit den Fokus bei Studierenden mehr auf Lernprozesse und gegenseitiges Helfen legen zu können. In mehreren durchgeführten Kursen an Universitäten beobachteten Schneider und Johnston, dass es immer Studierende gab, die für die weniger technischen Aspekte der Projekte nicht zu interessieren waren. Gerade in diesen Fällen stellten sie jedoch fest, dass es diesen Personen schwer fällt, Ideen und Lösungen zu kommunizieren oder ein Gefühl dafür zu entwickeln, dass die anderen Teammitglieder nicht die selben Denkstrukturen aufweisen und es so eigentlich von Nöten wäre, gemeinsame Ziele und Wege zu formulieren, um ein gutes Gesamtprodukt zu entwickeln. Bei diesen Studierenden sehen sie trotz handwerklich guter Fähigkeiten die Gefahr, dass ihnen wichtige Kompetenzen für ihre spätere berufliche Laufbahn in der professionellen Softwareentwicklung fehlen werden[2]. Für Schneider und Johnston ist ein entscheidendes Ziel, dass Programmierer lernen müssen, dass die Softwareentwicklung fast nie eine Aktivität einer einzelnen Person darstellt. Aus diesem Grund schlussfolgern sie, dass die agilen Methoden für den Einsatz in der Lehre als Ganzes durchaus reizvoll sind. Sie sind jedoch pessimistisch bezüglich der Anwendbarkeit, da sie zu viele Randbedingungen erkennen, die gegenläufige Ideale und Herangehensweisen unter Studierenden und Lehrenden fördern und so einen hohen Widerstand gegenüber den agilen Methoden hervorrufen. Die Autoren gehen daher davon aus, dass an der Verwendung einzelner Praktiken der agilen Methoden festzuhalten ist (z.B. *pair programming*), um zumindest wichtige Teilaspekte, wie die explizite Förderung der Kommunikationsfähigkeiten, abzudecken.

[2]Hier findet sich also wieder das Bild des Solisten in der Informatik, das dem gesellschaftlichen Ansehen aus Kapitel 2 entspricht.

McKinney und Denton [2005] berichten ebenfalls vom Einsatz einiger Praktiken der agilen Methoden (*pair programming*, *collective code ownership*, *on-site customer*, *stand up meetings*, *test-driven development*). Ihnen war es möglich, durch *Peer Evaluationen* bestimmte Handlungsweisen der einzelnen Teilnehmer in Bezug auf Aspekte wie Kommunikation und Kooperation zu identifizieren. Sie stellen dabei drei Kategorien vor, die gutes, schlechtes und absichtlich unkooperatives Verhalten der Teilnehmer beschreiben. Sie schlagen zur Eindämmung der beiden letzten Kategorien vor, klar definierten Anforderungen an die Gruppenteilnahme bereitzustellen und konstantes Feedback hierzu zu geben. Darüber hinaus empfehlen McKinney und Denton bei unkooperativem Verhalten Einzelner maßregelnd einzuschreiten und bei fehlendem Verständnis dafür die Paare aufzulösen, um die Verantwortlichen zu separieren.

Ein etwas anderes Ziel verfolgen Keefe et al. [2006]. Sie sehen die Verwendung von vier Praktiken aus XP (*pair programming*, *test-driven development*, *simple design*, *refactoring*) als Möglichkeit, um die pädagogische Qualität von Einführungsveranstaltungen anzuheben. Besonders erwähnen sie dabei die Bezüge zur realen Berufswelt und hoffen so auf das spätere Berufsleben besser vorbereiten zu können. Anhand einer Fallstudie arbeiten Keefe et al. mehrere Praxistipps für die Praktiken heraus. Für das *pair programming* empfehlen sie zu Anfang des Kurses die Paareinteilung den Studierenden zu überlassen, um ihnen eine sanfte Eingewöhnungsphase mit ähnlich motivierten Partnern zu ermöglichen.

### Pair Programming

Sucht man herausgelöste Komponenten des Projektmanagements der agilen Methoden in der Informatikbildung, so ist *pair programming* die einzige Praktik, die vielfach besprochen und positiv erwähnt wird (z.B.[Berenson et al., 2004; Burnett et al., 2010; McDowell et al., 2003; Mendes et al., 2005; Nagappan et al., 2003; Simon und Hanks, 2008; Williams und Upchurch, 2001]). Es wird berichtet, dass *pair programming* den Lernprozess fördert und Studierenden dabei hilft, bessere Quelltexte zu schreiben [Braught et al., 2011; McDowell et al., 2003; Mendes et al., 2005]. Zusätzlich findet sich häufig die Aussage, dass *pair programming* Einführungsveranstaltungen verbessert, indem es positive Erlebnisse und Erfahrungen unter Studierenden schafft und ihnen so das Studium vereinfacht [Braught et al., 2011; Nagappan et al., 2003; Simon und Hanks, 2008; Williams und Upchurch, 2001]. Darüber hinaus wird immer wieder hervorgehoben, dass gerade weibliche Teilnehmer vom *pair program-*

*ming* profitieren, da es soziale Aspekte der Entwicklungsprozesse hervorhebt und ihnen verdeutlicht, dass die Disziplin für sie eine gute Zukunftsperspektive darstellt [Berenson et al., 2004; Burnett et al., 2010; Werner et al., 2004].

Stellvertretend seien hier zwei Arbeiten im Detail beschrieben, da sie die oben genannten Punkte gut verdeutlichen. Gestützt auf mehrere Studien beschreiben Braught et al. [2011] *pair programming* als eine Praktik, die es vermag bessere Programmierfähigkeiten zu vermitteln – gepaart mit der Stärkung des Bewusstseins für ein allgemeines Informatikverständnis. Zusätzlich heben sie hervor, dass *pair programming* eine angenehmere Lernatmosphäre für die Studierenden erzeugt. Mit den Aussagen von Werner et al. [2004] lassen sich die positiven Aspekte, die Braught et al. [2011] hervorheben, um den Gesichtspunkt ergänzen, dass Frauen in gesondertem Maße von diesen Aspekten profitieren. Werner et al. [2004] beschreiben eine Studie, in der sie Einführungskurse mit verschiedenen Vorgehensmodellen (*pair programming* – Einzelarbeit) untereinander verglichen haben. Sie stellen fest, dass der Prozentsatz an Frauen, die die Kursteile erfolgreich abschließen, beim *pair programming* größer war als beim Kursteil, der Einzelarbeit voraussetzte. Außerdem waren sie in der Lage zu zeigen, dass Frauen, die zu Beginn des Studiums mit *pair programming* in Kontakt kamen, sich eher für Informatik-Abschlüsse entschließen und sich folglich für ein Informatik-Studium entscheiden[3]. Des Weiteren stellten Werner et al. fest, dass die Frauen aus *pair programming* ein deutlich gestärktes Selbstbewusstsein bezüglich ihrer Informatik-Kenntnisse ziehen konnten.

### Stichprobe Universität Hamburg

Es handelt sich bei den oben genannten Literaturquellen natürlich um eine internationale Auswahl, bei der zu berücksichtigen ist, dass die Bildungskonzepte in verschiedenen Ländern unterschiedlich gestaltet sind. Somit ist es fraglich, wie transferierbar die hier zusammengetragenen positiven Erkenntnisse auf einen deutschen Bildungskontext sind. Um den Eindruck zu stärken, dass es sich hierbei durchaus um ein übertragbares Phänomen handelt, sei hier kurz auf eine Umfrage verwiesen, die der Autor gemeinsam mit Dr. Axel Schmolitzky durchgeführt hat. Die Befragung wurde im Rahmen der Erstsemesterveranstaltung *Software Entwicklung 1* am Fachbereich Informatik der Universität Hamburg durchgeführt. Ziel war es, das in der Veranstaltung weitläufig eingesetzte *pair programming* in diesem konkreten Fall zu

[3]Wie unschwer zu erkennen ist, handelt es sich hier um einen Artikel aus den USA, wo *undergraduates* technische Einführungsveranstaltungen besuchen können und sich, wenn angestrebt, erst später für die Ausrichtung ihres *majors* entscheiden.

hinterfragen und die entsprechenden Eindrücke der Studierenden dazu in Erfahrung zu bringen. An der Umfrage nahmen 254 Personen teil. Der Fragebogen (siehe A.1) sowie die durch das Evaluationswerkzeug automatisch generierte Auswertung (siehe A.2) sind im Anhang zu finden.

Es wurde bei der Befragung deutlich, dass die Mehrheit der Befragten vor dem Studium noch nie etwas von *pair programming* in der Softwareentwicklung gehört hatte (62,8%, n=250). Auf die Frage, ob *pair programming* das Gemeinschaftsgefühl unter den Teilnehmern stärkt, antwortete eine große Mehrheit mit *ja* (79,1%, n=244). Auf die Frage, ob *pair programming* das Verständnis für die Inhalte der Veranstaltung fördere, waren 50,8% der Ansicht, dass es ihnen sehr helfe, 43,2% betrachteten es neutral, da sie es nicht zu beurteilen vermochten und 6% antworteten, dass sie es nicht für hilfreich befänden (n=250). Bei dieser Frage konnte eine signifikante Abhängigkeit (0,05-Level) vom Geschlecht festgestellt werden: Die Teilnehmerinnen bewerteten bei dieser Frage *pair programming* deutlich hilfreicher für das Verständnis des Stoffs als männliche Teilnehmer.

Diese Zahlen reichen aus, um zu argumentieren, dass die Erkenntnisse aus internationalen Beobachtungen auch im deutschen Bildungssystem zutreffend sind und demnach *pair programming* eine Praktik ist, die Anfängern zu besseren Lernerfahrungen und mehr Selbstbewusstsein verhilft. Auch soziale Komponenten können anscheinend gut durch *pair programming* hervorgehoben werden. Es ist zu vermuten, dass diese positiven Eigenschaften noch stärker in Verbindung mit Softwareentwicklung oder Informatik in Verbindung gebracht würden, wenn bereits vor der Studienwahl die Bekanntheit dieser Methoden gesteigert werden könnte. Für einen zu geringen Bekanntheitsgrad spricht, dass sogar 62,8% der Informatik-Studierenden vorher noch nie etwas von *pair programming* gehört hatten. Es ist folgerichtig anzunehmen, dass dieser Anteil bei anderen Gruppen, die sich nicht für ein Informatik-Studium entschieden haben, noch größer ist.

## 3.2 Informatik an Schulen

Im oberen Kapitelabschnitt sollte klar geworden sein, dass an Universitäten agile Methoden in der Informatikbildung zunehmend als starkes Instrument wahrgenommen werden, aber noch zu selten eingesetzt werden. So entsteht erst verzögert die Chance, weitläufig im universitären Bereich auf soziale attraktive Lernumgebungen und Berufsaussichten der Informatik hinzuweisen. Schüler könnten dann höchstens von älteren Bekannten oder Verwandten, die sich für ein Informatikstudium entschie-

den haben, von agilen Methoden bzw. sozialen Aspekten in der Softwareentwicklung berichtet bekommen. Es steht daher zu befürchten, dass die universitären Umsetzungen der agilen Methoden und mit dem entsprechenden Fokus auf soziale Interaktion deutlich zu spät, nämlich nicht im Schulkontext, umgesetzt werden, um unter Jugendlichen Bekanntheit zu erlangen. Der folgende Abschnitt gibt daher einen Einblick, wie im Schulkontext Jugendliche an das Programmieren herangeführt werden und ob vielleicht dort die Möglichkeit besteht, soziale Prozesse in der Informatik bzw. Softwareentwicklung selbst zu erfahren. Dies geschieht, indem betrachtet wird, wie an Schulen Informatik laut Stand der Wissenschaft vermittelt werden soll. Zum einen werden im Folgenden Empfehlungen und wissenschaftliche Arbeiten vorgestellt. Zum anderen wird eine Befragung präsentiert, die unter 17 Informatiklehrern aus dem Raum Hamburg und Schleswig-Holstein durchgeführt wurde (siehe dazu Kapitel 3.2.2), um ein Gefühl zu erlangen, wie dies in der Praxis umgesetzt wird.

Zahlreiche Beobachtungen und Gespräche sowohl mit Schülern als auch mit Lehrern im Rahmen von Projektveranstaltungen (im Zeitraum 2007–2011) ergaben, dass es gerade im Informatikunterricht an Methoden und Ideen mangelt, die einen außerschulischen Realitätsbezug haben, um darüber hinaus soziale Interaktion als Kernelement der Softwareentwicklung vorstellen zu können. Bei einer allgemeinen Betrachtung zu Gruppenarbeit im Schulkontext von Antil et al. [1998] stellte sich heraus, dass beispielsweise über 95% der Teilnehmer einer Studie mit Lehrern angaben, Gruppenarbeit zu verwenden. Gleichzeitig hatten von diesen Personen zwei Drittel eigene Variationen von Gruppenarbeit entwickelt und setzten diese ausschließlich ein. So ist ersichtlich, dass Schüler solche Methoden als schul- oder lehrerspezifisch ansehen werden und keine Referenzen zu realen Berufsbildern herstellen können, was jedoch durchaus möglich und besonders im Kontext der Softwareentwicklung aussichtsreich wäre.

Die folgenden Betrachtungen bis Seite 39 sind Göttel [2011a] entnommen, der Wortlaut wurde nur in geringem Maße angepasst.

Es wird davon ausgegangen, dass bereits in Empfehlungen und Bildungsstandards Elemente enthalten sein sollten, die zur Steigerung der Attraktivität des Fachs beitragen und konkret einsetzbar sind. Ein Fokus auf die reine Zweckmäßigkeit und die großen Berufschancen dürfte nicht ausreichend sein, um IT-Berufe in ein besseres und zutreffenderes Licht zu stellen.

In den Grundsätzen und Bildungsstandards für die Informatik, die durch die GI herausgegeben wurden[4], finden sich einige Hinweise darauf, dass Informatik als ein Schulfach zu begreifen ist, das besondere interdisziplinäre und soziale Aspekte aufweist [Gesellschaft für Informatik e.V., 2008]. Hierbei fällt auf, dass besonders zu Anfang der Publikation darauf Wert gelegt wird, dass Lösungen für informatische Probleme häufig im Team erarbeitet werden müssen. Daher wird verlangt, dass ein guter Informatikunterricht den Schülern das Selbstvertrauen dazu vermittelt. Ein weiterer Aspekt ist, dass es den Verfassern darauf ankommt, dass die informatische Schulausbildung vorbereitet auf *„das Leben, so wie es ist“*. Die Autoren siedeln ihre Empfehlungen bewusst zwischen Input- und Output-Orientierung[5] an, so dass es kaum nachvollziehbar ist, dass Hinweise zu passenden Methoden zur Vermittlung von sozialen Aspekten nicht auffindbar sind. Es ist verwunderlich, dass immer wieder darauf hingewiesen wird, dass künftige Lebenssituationen (a.a.O., S.1, 4, 6) berücksichtigt werden sollen, aber nie darauf verwiesen wird, dass es durchaus Möglichkeiten gibt, sich der Elemente der Softwareentwicklung zu bedienen, um dies realitätsnah zu gestalten. Auch wird darauf eingegangen, dass das Erarbeiten von informatischem Wissen gemeinsam stattfinden soll. Begründet wird dies jedoch meist damit, dass so möglichst effektiv Informatikstoffe vermittelt werden können; auch hier finden sich keine Anhaltspunkte dafür, dass der Bezug zur realen Softwareentwicklung die Attraktivität des Fachs steigern könnte. Der Artikel stellt heraus, dass nur in der Informatik die Möglichkeit für Schüler besteht, zu erfahren, *„ob die konstruktive Arbeit mit technischen Werkzeugen für sie möglich und attraktiv, eben eine Lebensperspektive ist“* und somit auch die reale Softwareentwicklung als ein wesentlicher Bestandteil an Schulen referenziert werden muss. Die Interdisziplinarität der Informatik wird darüber hinaus erwähnt, jedoch nicht als Faktor der Informatik erkannt, der zur Attraktivitätssteigerung beitragen könnte; vielmehr wird nur darauf verwiesen, dass so fächerübergreifende Stoffe vermittelt werden können. Hier wäre eine offensivere Herangehensweise wünschenswert, die es vermag, gerade den von Haus aus an der Informatik zweifelnden, aber interessierten Schülern das Fach schmackhaft zu machen. Eine genauere Untersuchung nach sozialen Aspekten lohnt sich bei den folgenden identifizierten Prozessbereichen: „Modellieren und Implementieren“, „Kommunizieren und Kooperieren“ und „Darstellen und Interpre-

[4]Die GI spricht hierbei von einem Mindestmaß an informatischem Wissen, das spätestens mit der mittleren Reife erlangt werden soll und somit breiten Anklang im zukünftigen Gesellschaftsbild finden soll.

[5]Bei einer Input-Orientierung werden konkrete Inhalte und Werkzeuge vorgegeben, wohingegen bei einer Output-Orientierung ausschließlich die zu erlernenden Fähigkeiten beschrieben werden.

tieren“. Auffallend bei „Modellieren und Implementieren“ ist, dass ein deutlicher Fokus auf Werkzeuge gelegt wird und dabei nicht erwähnt wird, dass gerade diese Arbeit nur noch selten als Einzelleistung in der Softwareentwicklung zu verstehen ist. Hier scheint die Möglichkeit vertan, den Schülern zu vermitteln, dass gerade das Erarbeiten dieser Lösungen einen hochgradig kreativen und zwischenmenschlichen Aspekt der IT-Berufsbilder darstellt. Insbesondere die Forderungen bei „Kommunizieren und Kooperieren“ erscheinen als treffend. Betrachtet man jedoch die Praxis, so scheint es einen Mangel an Methoden zu geben, die Lehrern an die Hand gegeben werden können (siehe hierzu nochmals 3.2.2). In den Betrachtungen zu „Darstellen und Interpretieren“ werden leider nur selten Hinweise gegeben, dass gerade diese Fähigkeiten eigentlich nur in Kleingruppen erlernt werden können, um so z.B. zu erfahren, wie Darstellungsformen von anderen Teilnehmern interpretiert werden.

In den Empfehlungen für ein Gesamtkonzept zur informatischen Bildung an allgemein bildenden Schulen, herausgegeben durch die GI, wird darüber hinaus auch die Vermittlung und Hervorhebung von Sozialkompetenz in den Aufgabenbereich der Informatikbildung gerückt. Es wird hierbei gezielt gefordert, dass Schüler befähigt werden sollen, *„Gruppenprozesse zu planen und mitzugestalten, Kritik entgegenzunehmen bzw. konstruktiv formulieren zu können, einen Arbeitsrollenwechsel zu erleben und akzeptieren zu können“* [Gesellschaft für Informatik e.V., 2000]. In diesem Text findet sich sogar der konkrete Hinweis, dass diese Fähigkeiten auch in der realen (IT-)Arbeitswelt wiederzufinden sind. Es wird jedoch nicht darauf verwiesen, dass diese Gegebenheit selbst auch ein zu vermittelndes Wissen ist.

In der Empfehlung der Kultusministerkonferenz (KMK) zur Stärkung der mathematisch-naturwissenschaftlich-technischen Bildung finden sich manche Forderungen, die in ihrer Interpretation und Durchführung zum Ansehen der Informatik und auch der Qualitätssicherung im Informatikunterricht beitragen können [Kultusministerkonferenz (KMK), 2009]. Darin wird darauf verwiesen, dass möglichst *„konkrete Erfahrungen mit naturwissenschaftlichen, ingenieurwissenschaftlichen und technischen Berufen zu ermöglichen“* sind. Hierbei wird in erster Linie darauf Wert gelegt, dass Netzwerke zu Firmen entstehen sollen, so dass sich Jugendliche vor Ort davon ein Bild machen können. Im Umkehrschluss ist jedoch auch denkbar, dass konkrete Methoden aus dem Berufsbild in den Schulkontext überführt werden sollten. Weiteren Anreiz bietet auch das in der Empfehlung der KMK beschriebene Handlungsfeld „Gesellschaftliche Akzeptanz“, in dem die Rede davon ist, dass das Interesse an technischen Fragestellungen bei Schülern zu wecken sei. Hier ist es natürlich ver-

wunderlich, dass nicht auch darauf hingewiesen wird, welche kreativen, sozialen und interdisziplinären Berufsbilder in diesem Bereich vorhanden sind.

Es existiert jedoch ein englischsprachiger Artikel von Meerbaum-Salant und Hazzan [2010], der diesen Bezug herstellt, indem er agile Methoden in einem Schulkontext beschreibt – wenn auch ausschließlich aus der Lehrersicht. Meerbaum-Salant und Hazzan versuchen High School Lehrern (Sekundarstufe) eine Methodologie zur Betreuung von Softwareprojekten an die Hand zu geben, die sich unter anderem auf agile Methoden stützt. Auch Meerbaum-Salant und Hazzan erkennen die in Kapitel 3.1.1 bereits genannten Werte der agilen Methoden als einen Anknüpfungspunkt für schulische Konzepte. Sie sehen diese Werte auch in dem *Teacher Knowledge Base Model*[6] von Schulman [1987] und dem Konstruktionismus nach Papert [1980]. Diese Quellen ziehen Meerbaum-Salant und Hazzan [2010] neben den agilen Methoden (um genauer zu sein XP) zur Bildung ihrer Methodologie heran. Die Methodologie bietet mehrere Praktiken an, um Lehrern zu erleichtern, den besonderen Anforderungen einer Projektbetreuung in einer Sekundarstufe gerecht zu werden. Die Praktiken leiten sie aus den genannten drei Quellen ab und unterteilen sie in drei Aspekte: Soziale Aspekte, Projektmanagementaspekte und pädagogische Aspekte. Für die einzelnen Praktiken stellen sie dazu einen exemplarischen Ablaufplan vor, der es Lehrern erlaubt über ein gesamtes Schuljahr zu planen. Es bleibt noch zu erwähnen, dass Meerbaum-Salant und Hazzan mit den agilen Methoden keinerlei soziale Aspekte abzudecken versuchen. Sie sprechen lediglich von Projektmanagement-Aspekten. Unter Berücksichtigung der in Kapitel 3.1.1 genannten positiven sozialen Eigenschaften von Praktiken der agilen Methoden in der universitären Informatikbildung scheint hier die Methodologie ein wenig zu kurz zu greifen. Da in dem Artikel nicht die Perspektive der Schüler eingenommen wird, scheint es auch nicht möglich, Überlegungen anzustellen, welche Außenwirkung Kurse haben sollen, die der Methodologie folgen. Somit ist fraglich, ob soziale Aspekte im Zusammenhang mit der Softwareentwicklung dadurch deutlicher hervorgehoben werden können. Kritisiert werden kann auch, dass der Artikel zwar wichtige Aspekte abdeckt, jedoch für die einzelnen Aspekte keine konkreten Abläufe beschreibt, die Lehrer benötigen, um diese Aspekte erfolgreich und in einem vertretbaren Aufwand im eigenen Unterricht zu vermitteln. Anscheinend gehen Meerbaum-Salant und Hazzan davon aus, dass die genannten Praktiken von Lehrern anhand der zitierten Quellen selbst erarbeitet

[6]Schulman [1987] unterscheidet sieben Kategorien, die ein Lehrer beherrschen und zusammenführen muss: Inhaltswissen, pädagogisches Grundwissen, Kenntnisse über das Curriculum, pädagogisches Fachwissen, Verständnis von der Zielgruppe, Bildungskontexte und Kenntnisse über Ziele und Einschränkungen des Unterrichts.

werden sollen, bzw. zu dem Thema angebotene Workshops besucht werden müssen. Es steht außerdem zu befürchten, dass aufgrund der verschiedenen Schulsysteme[7] englischsprachige Artikel ohne konkreten deutschen Bezug nur eine geringe Verbreitung im deutschsprachigen Raum besitzen.

Etwas mehr Hoffnung auf eine angemessene Verbreitung darf man somit auf den deutschsprachigen Artikel von Weigend [2005] setzen. Dieser legt dar, wie man Schüler durch programmiertechnische Praktiken aus dem XP zu qualitativ besseren Programmierergebnissen verhelfen kann. Weigend sieht XP als sinnvoll für den Unterricht an, in diesem Zusammenhang hebt er hervor, dass durch die mehrmaligen Iterationen sicher gestellt ist, dass jedes Team ein präsentierbares Ergebnis erhält. So kann von einem Erfolg gesprochen werden, auch wenn nicht zwangsläufig alle ursprünglich definierten Ziele erreicht wurden. Darüber hinaus erwähnt er, dass Paare sich die Zeit und Ziele selbst einteilen müssen und somit die Möglichkeit für selbstständiges Experimentieren und Forschen gegeben ist. Daraufhin geht Weigend auf die von ihm vorgeschlagenen XP-Praktiken und Abläufe zum Gebrauch im Schulunterricht ein: *Release planning* mit *stories*, *spikes*, Ausführung einer Iteration mit *test driven development* und *refactoring* sowie *big visible charts* zur Dokumentation. Weigend ist in all seinen Ausführungen zu den Praktiken darauf fokussiert, herauszuarbeiten, dass Schüler damit befähigt werden, durch eigene gestaltete, aber klar vordefinierte Abläufe, einen Entwicklungsprozess zu durchlaufen. Dies soll zum einen das Lernen durch den konstruktiven Charakter erleichtern und zum anderen qualitativ bessere Programme als sonst im Informatikunterricht entstehen lassen. Es ist jedoch verwunderlich, dass Weigend sowohl bei der Verwendung von *stories* als auch bei den *big visible charts* (die natürlich auch als *informative workspace* gesehen werden können) nur beiläufig bzw. implizit erwähnt, dass den Teilnehmern auf diese Weise auch ein Austausch untereinander erleichtert wird. Die vorliegende Arbeit dagegen ist in der Überzeugung geschrieben, dass dieser positive Aspekt des sozialen Austauschs aufgewertet werden muss, indem es neben den von Weigend genannten Vorteilen von XP als weiteres wichtiges Ziel definiert wird.

Betrachtet man die genannten Empfehlungen, Hinweise und Artikel, so ist nachvollziehbar, dass es mangels konkreter Praktiken und häufig anders gesetzten Zielen schwierig für Lehrer ist, sozialorientierten Inhalte als Ziel wahrzunehmen und/oder angemessen zu vermitteln. Darüber hinaus scheint es auch verpasst worden zu sein,

[7] Es ist daran zu erinnern, dass bereits in Deutschland starke Unterschiede in der Bildungsstrukturen existieren, da die Entscheidungskompetenz auf Bundesländerebene liegt. Transnationale Erkenntnisse sind daher selten zu erwarten oder zu beachten.

diese sozialen Aspekte als konkreten Anknüpfungspunkt zum Berufsbild der IT wahrzunehmen und als solches darauf hinzuweisen. Diese Aspekte werden somit nicht oder nur selten ihren Weg in die Schulpraxis finden. Um diesen Eindruck zu überprüfen, wird nachfolgend in Kapitel 3.2.1 ein Überblick zu Kooperation und sozialen Aspekten in computergestützten Lernumgebungen und im deutschsprachigen Informatikunterricht gegeben. Abschließend werden dann die Ergebnisse einer Informatiklehrerbefragung in Kapitel 3.2.2 herangezogen, um den entstandenen Eindruck mit einer Stichprobe zu prüfen.

### 3.2.1 Computergestützte Gruppenarbeit und Kooperation

Zunächst wird der Begriff der Gruppe im Bildungskontext allgemein dargestellt. Hierfür wird auf Arbeiten aus dem *computer supported cooperative learning* (CSCL) bzw. der CGK zurückgegriffen. Eine soziale Gruppe erkennt man nach Janneck und Janneck [2004] an einer ständigen Kommunikationsmöglichkeit, an einer Abgrenzung zur Umwelt über eigene Strukturen, an einem Gemeinschaftsgefühl der Teilnehmer und an der Zusammenarbeit bzw. an der wechselseitigen Unterstützung der anderen Gruppenmitglieder. Ebenfalls nach Janneck und Janneck sollte im Lernkontext eine gleichberechtigte Gruppenstruktur sicher gestellt werden, wobei Konkurrenz unter den Teilnehmern zu vermeiden ist. Sowohl Janneck und Janneck als auch Johansen et al. [1991] sprechen von einem Lebenszyklus von Gruppen, den es zu unterstützen gilt. Johansen et al. sehen in der CGK sieben Phasen, die sich in zwei Kategorien, die Aufbauphasen und die Erhaltungsphasen, einteilen lassen und mit verschiedenen technischen als auch organisatorischen Maßnahmen unterstützt werden müssen.

Wessner [2004] führt an, dass bei Lerngruppen soziale Interaktion essentiell ist, um das Lernen bzw. den Wissenserwerb der Gruppenmitglieder zu fördern. Wessner lobt insbesondere neben dem eigentlichen Produktergebnis einer Lerngruppe die Ausbildung von „Sozial-, Fach- und Methodenkompetenz“ dieser Lernform. Einen essentiellen Bestandteil sieht er in der Koordination der Aufgaben und Aktivitäten innerhalb einer Lerngruppe. Hier stimmt er mit Janneck [2004] überein, der Projektorientierung als Lernform im Zusammenhang mit Lerngruppen sieht und besonders die Koordination bzw. „kooperative Planung“ technisch unterstützt sehen will. Diese Empfehlungen aus der CSCL sollen im Rahmen dieser Arbeit als weitere Begründung ausreichen, um einen Fokus auf Lerngruppen mit gemeinsamen Zielen zu legen, deren Teilnehmer in ihrem Handeln und Planen sowohl technisch als auch methodisch-didaktisch unterstützt werden sollen. Diese Szenarien werden also

durch ihren sozialen Charakter nicht nur als attraktiv wahrgenommen, sondern sie sind auch als Lernform erfolgreich.

Während es sich bei den oben genannten Quellen um allgemeine Betrachtungen zum Lernen und Arbeiten in Gruppen handelt, gilt es nun wieder den Schulkontext aufzugreifen. Da es sich bei schulischer Bildung zumeist um differente nationale, aber bindende Randbedingungen handelt, erweist es sich als sinnvoll, deutschsprachige Fachartikel der Didaktik der Informatik mit vereinendem Charakter hinzu zu ziehen. Oftmals finden sich bereits auf Länderebene starke formale Unterschiede bei der Informatikbildung in Kursform, Art, Dauer und Inhalten (vergleiche hierzu [Starruß, 2010]. Aus diesem Grunde wird folgend exemplarisch auf das Buch „Didaktik der Informatik“ von Schubert und Schwill [2011] und die deutschsprachige Zeitschrift *LOG IN* zurückgegriffen, die Artikel zur Informatikbildung an Schulen publiziert und bundesweit relevante Themen aufgreift.

Schubert und Schwill beschreiben Unterrichtsformen in der Schulinformatik und entsprechende mögliche Kooperationsszenarien [Schubert und Schwill, 2011, S. 304]. Zum einen unterscheiden sie bezüglich der Sozialform nach „Unterricht im Klassenverband“, „Gruppenunterricht“ und „Einzelunterricht“. Die verlangten und auftretenden Aktivitäten der Teilnehmer innerhalb dieser Szenarien fassen sie wie folgt zusammen: „kommunizierende Form“, „gelenktes Entdecken“ und „freies Forschen“. Eine Unterrichtsform ist immer eine Kombination aus jeweils mindestens einem Merkmal der Sozialformen und der Aktivitäten. Für die Informatikbildung empfehlen sie einen Projektunterricht, den man als eine Mischform aus „Unterricht im Klassenverband“, „Gruppenunterricht“, „gelenktes Entdecken“ und „freies Forschen“ begreifen kann. Konkrete Hinweise, wie der Gruppenunterricht zu gestalten sei, geben Schubert und Schwill nicht. Sie stellen hingegen informatische Aspekte des Projektunterrichts vor, die sich am Wasserfallmodell (vergleiche Kapitel 2.2) orientieren und geben damit implizit Anknüpfungspunkte für die Gestaltung des Projektunterrichts vor [Schubert und Schwill, 2011, S. 308 ff.]. Das Wasserfallmodell schließt zwar Gruppenarbeit nicht aus[8], durch seine Linearität kann jedoch recht einfach der Eindruck entstehen, dass es sich um eine Aufgabe handelt, die eine Person auch alleine durchlaufen kann oder sogar soll. Diese Schlussfolgerung stärkt wiederum das Bild der Solisten-Disziplin aus Kapitel 2.3.1.

Als Grundlage für die folgenden Betrachtungen wurden alle Ausgaben seit Erscheinen der Zeitschrift *LOG IN* (1981–2011) auf relevante Artikel geprüft, indem

[8]So gibt es beispielsweise auch modifizierte Wasserfallmodelle zum Einsatz im Schulunterricht, die Iterationen vorsehen [Thomas, 2002, S. 75].

sämtliche Titel und Untertitel auf die Suchworte *Kooperation/kooperativ*, *Kollaboration/kollaborativ*, *Team*, *Gruppe* untersucht wurden[9]. Auf diese Weise wurden elf Artikel identifiziert, die sich augenscheinlich mit der Thematik befassen. Nachfolgend werden die neun Artikel aus dieser Auswahl vorgestellt, die sich als brauchbar für die vorliegende Diskussion erwiesen.

Rauch [1994] fordert, dass sich didaktische und methodische Herangehensweisen im Unterricht allgemein unter dem Gebrauch von computergestützter Kommunikation und Kooperation (in seinen Worten *groupware*) verändern sollen, um dazu beizutragen, dass Teamfähigkeit zu einem fächerübergreifenden Lernziel bestimmt werden kann. Rauch formuliert, dass Inhalte derart aufbereitet werden müssen, damit sie *groupware*-Aspekte und gemeinsames Arbeiten hervorheben. Wohlgemerkt handelt es sich um einen Artikel aus der Zeit, als *Microsoft Windows for Workgroups* auf den Markt kam; ein System, das erste Schritte hin zu vernetzten Rechnerstrukturen im privaten Anwendungsbereich schuf, jedoch bei weitem nicht die umfassenden Kommunikationsformen anbot, die zur heutigen Zeit weit verbreitet sind (z.B. *E-Mail*, *instant messaging*, *shared spaces*, etc.)

Lehmann [1996] stellt sich die Frage, wie man mit Teilnehmern umgehen soll, die eindeutig ein umfangreicheres Wissen über das Programmieren haben als die anderen Teilnehmer, und wie man diesen in Gruppenarbeit gerecht werden kann. Neben anderen Fertigkeiten kommt es Lehmann in der Informatikbildung auch auf Kommunikationsfähigkeit und die Tauglichkeit zur Gruppenarbeit an. Genau da macht er auch bei „Informatik-Freaks“ Schwachstellen aus und empfiehlt, hierauf abzielende Extraaufgaben zu verteilen. So schlägt er beispielsweise vor, sie als „Hilfslehrer“ oder Projektmanager einzusetzen, um das Wissen dieser Personen anderen Gruppenmitgliedern zugänglich zu machen und gleichzeitig das Vermitteln von Wissen zu schulen.

Schmitz und Thiele [1999] stellen Schülerprojekte vor, die in Kooperation mit einer Universität betreut wurden. In dieser projektbezogenen Teamarbeit stellten die Autoren fest, dass so das Interesse für zugrunde liegende mathematische Fragestellungen wächst und darüber hinaus auch soziale Kontakte über den Klassenverband hinaus entstehen können. Auch konnten Schmitz und Thiele beobachten, dass durch eine selbständige Aufgabenverteilung innerhalb der Gruppen und die offene Diskussionsatmosphäre zwischen allen Beteiligten (Schüler und Betreuer) das Selbstbewusstsein der Teilnehmer gesteigert werden konnte.

[9]Eine Volltextsuche ist leider für die *LOG IN* nicht verfügbar.

Auch Humbert [1999] stellt unter Bezug auf weitere Literaturquellen heraus, dass der Computer als ein soziales Medium aufzufassen ist und als solches auch im Unterricht an Schulen genutzt werden sollte. Humbert macht in erster Linie die rasche und damit preisgünstige technische Entwicklung dafür verantwortlich, dass gerade in Schulen die Forderung besteht, dass jeder Schüler auf einen eigenen Computer zugreifen können soll und demnach auch Aufgaben strikt alleine zu bearbeiten haben. Ziel des im Artikel vorgestellten Schulunterrichts war es, Metaphern und Vorgehen der CGK zu vermitteln, also bewusst ohne konkrete Bezüge zum Programmieren, zur Softwareentwicklung bzw. zur Informatik. Der einzige Bezug wird dazu hergestellt, indem darauf verwiesen wird, dass die verwendeten CKG-Systeme in einer objektorientierten Programmiersprache entwickelt wurden. Dabei wird nicht erwähnt, dass die sozialorientierten Aspekte des CKG sicherlich auch in großen Maßen in der Entwicklung der Systeme zu finden sind.

Rüdiger [1999] erkennt in den immer breiter verfügbaren CKG-Technologien zur damaligen Zeit die Möglichkeit, die Handhabung aber auch die Grundlagen für gut strukturierte Gruppenarbeit im Informatikunterricht zu thematisieren. Im traditionellen Unterricht sieht die Autorin durch die Gemeinsamkeit von Ort und Zeit soziale und kommunikative Aspekte zu wenig berücksichtigt, als dass sie auf die spätere Arbeitswelt vorbereitend ist, die zunehmend auf Gruppen an verteilten Orten und zu verschiedenen Zeiten setzt oder setzen muss. Für Rüdiger liegt es auf der Hand, dass Gruppenarbeit im Kontext von CKG-Systemen eine stärkere Bedeutung im Informatikunterricht spielen muss. Um dies zu untermauern, berichtet sie von einer Studie mit 59 Schülern eines beruflichen Gymnasiums, die hervorbrachte, dass nur eine geringe Minderheit dieser Teilnehmer vorher Gruppen- und Projektarbeit in Schulen erlebt hatte. Vorwerfen kann man Rüdiger jedoch, dass sie kaum konkrete Abläufe oder Arbeitsschritte vorstellt und statt dessen im zweiten Teil des Artikels systemorientiert auf die vorhandenen CKG-Lösungen eingeht. So steht zu befürchten, dass es ihr in erster Linie darum geht, diese Systeme als solches zu vermitteln und sie so möglicherweise die Aspekte der Gruppendynamik nicht mehr ausreichend beleuchtet.

Janneck [2006] dagegen wählt einen ganz anderen Ansatz: Er sieht Schüler erst einmal als Nutzer, die als solches auch partizipativ in die Systementwicklung einbezogen werden sollen – möglichst ohne dass sie einen Rollenwechsel vom Nutzer zum Programmierer vollziehen müssen (oder möglichst spät). Er stellt heraus, dass grundlegende Strukturen so deutlich leichter zu erlernen sind, da syntaktische Details nicht so sehr ins Gewicht fallen. Janneck schlägt dazu – angelehnt an die pro-

fessionelle partizipative Softwareentwicklung – Papierprototypen und Szenarien als gut geeignete und erlernbare Werkzeuge vor. Gerade in Papierprototypen sieht er eine Chance, kooperatives Arbeiten zu propagieren. Darüber hinaus legt er Wert darauf, dass die Teilnehmer vor dem Erstellen der Prototypen in Interviews mit verschiedenen potentiellen Zielgruppen wichtige Eigenschaften für das zu erarbeitende System erfragen. Auch in diesem Ansatz werden also soziale Aspekte der Softwareentwicklung den Schüler frühzeitig zugänglich. Dieser Artikel ist mit den praktisch nachvollziehbaren Empfehlungen den Zielen der vorliegenden Arbeit zuträglich.

Ein Artikel von Homberg [2006] lobt die didaktischen Möglichkeiten von Lernplattformen am Beispiel *moodle*. Er unterstreicht, dass dort das „Experimentieren, das Recherchieren und das eigene Gestalten im sozialen Austausch mit Mitschülern“ unterstützt wird. Auch er sieht große Vorteile eines solchen Systems und das Potenzial, dass die sonst übliche Gleichheit von Ort und Zeit im Schulunterricht mit solchen Systemen durchbrochen werden kann. Es ist ihm gleichzeitig jedoch wichtig, dass solche Lernplattformen immer in einer Wechselwirkung zu einem Präsenz-Unterricht stehen müssen, um beide Herangehensweisen gewinnbringend zu verknüpfen und deren jeweiligen Nachteile aufzuheben. Als Unterrichtsbeispiel das mit *moodle* unterstützt wird, wählt Homberg das Thema Urteilsbildung und verpasst so, soziale Aspekte, die ein gemeinsames Lerngefühl vermitteln, im Kontext der Informatik zu etablieren.

Kohls und Haug [2008] und König [2008] befassen sich mit Wikisystemen im Schulkontext. Kohls und Haug [2008] sehen Wikis als universell einsetzbar und heben hervor, dass sie sich besonders für die Präsentation von Projektergebnissen und einen gemeinsamen Aufbau eines Wissensspeichers eignen. Für Kohls und Haug steht außer Frage, dass Wikis von den Nutzern jedoch eine „Bereitschaft zur Diskussion sowie ein gewisses Maß an Kritikfähigkeit erfordern“. Auch die Notwendigkeit von Vertrauen und Kommunikationsfähigkeit bei den Nutzern entdecken sie in Wikis. Auch wenn von Kohls und Haug nicht explizit genannt, rücken demzufolge soziale Komponenten automatisch ins Blickfeld des durch Wikis gestützten Unterrichts. Konkrete Unterrichtsvorschläge im Bezug zur Informatikbildung stellen sie nicht vor, sie geben einzig allgemeine Tipps, wie Wikis im Unterricht präsentiert werden sollten. König [2008] erwähnt darüber hinaus flache Hierarchien und Teamgedanken, die in Wikis ständig zum Tragen kommen. Nach Döbeli Honegger [2007] sieht König [2008] die didaktische Nutzung von Wikis als besonders geeignet, da sie Publikationsmöglichkeiten auch ohne HTML-Kenntnisse bieten und „wesentliche Potenziale zum Erwerb sozialer Kompetenzen“ besitzen. Sein Vorschlag für den Einsatz von

Wikis bezieht sich auf die Einbettung in Schulwebseiten und hat demzufolge keinen Informatikunterrichtsbezug.

Zusammenfassend lässt sich sagen, dass es einige Artikel gibt, die Computersysteme oder damit verbundene Prozesse als sozialorientierte Umgebungen begreifen, gleichzeitig aber häufig vernachlässigen aufzuzeigen, dass auch bei der Entwicklung von Computerprogrammen solche sozialen Aspekte vorhanden und essentiell für gute Entwicklungsprozesse sind. Eine Ausnahme stellt die Arbeit von Janneck [2006] dar, da dort Bezüge zu realen Softwareentwicklungsprozessen hergestellt werden, die soziale Komponenten beinhalten.

### 3.2.2 Lehrerbefragung

Der folgende Abschnitt zur Lehrerbefragung findet sich in großen Teilen bei Göttel [2011a] wieder.

In einem Fragebogen (siehe Anhang B) wurden – abzielend auf die oben beschriebenen sozialen Aspekte im Informatikunterricht – 17 Informatiklehrer aus Hamburg und Schleswig-Holstein befragt. 13 Teilnehmer waren Gymnasial-, zwei Real- und einer Gesamtschullehrer, insgesamt vier davon waren weiblich. Die Studie wurde im Anschluss an einen Workshop durchgeführt, die Teilnahme war freiwillig.

Auf die Frage, in welchen Veranstaltungsformen Informatikinhalte durch die befragte Person vermittelt wurden, ergab sich folgende Verteilung (Mehrfachnennungen waren erlaubt): drei als Pflichtfach, zwölf als Wahlpflicht, zwei Wahlfach, sechs AG und nur eine Projektwoche. Diese Antworten sind bemerkenswert, da die Mehrzahl in Wahlpflicht und Wahlfach liegt. Naheliegend ist, dass zu diesem Zeitpunkt bereits eine Vorauswahl der Schüler stattgefunden hat. So ist davon auszugehen, dass nur noch eine gewisse, ohnehin schon informatikaffine Klientel teilnimmt. Wünschenswert wäre eine Vielzahl von Projektangeboten, da dort meistens ein breiteres Spektrum an Schüler beteiligt ist und somit auch die Chance besteht, andere Zielgruppen für die Informatik zu begeistern. Darüber hinaus dürften gerade Projektwochen und AGs besonders gut den Projektcharakter der Softwareentwicklung vermitteln.

Bei der Frage, welche Inhalte vorwiegend in den Lerneinheiten programmiert wurden (Mehrfachnennungen waren auch hier erlaubt), gaben sieben Teilnehmer an, dass Anwendungen entwickelt wurden, sechs bezeichneten Algorithmen als Hauptfokus und niemand identifizierte Games als Lerninhalt. Darüber hinaus gab es noch andere Anwendungen, die Verwendung fanden und extra genannt wurden: PovRay,

Robotik/Lego Mindstorms, Robot Karol und Microsoft Office. Positiv bleibt festzuhalten, dass ein Fokus auf Anwendungen lag (die ja auch Algorithmen beinhalten können). So ist wenigstens davon auszugehen, dass prinzipiell im Informatikunterricht auf Fragen der Softwareentwicklung Bezug genommen werden kann. Unter dem Aspekt der Kreativität in der Informatik ist es jedoch enttäuschend, dass niemand die gute Gelegenheit nutzt und mit Games das aufgreift, was die meisten Jugendliche begeistern dürfte und darüber hinaus einen guten Anknüpfungspunkt an aktuelle Entwicklungen in der Softwareentwicklung darstellt.

In der Befragung zu verwendeten Web-Plattformen zur Unterstützung der Arbeit der Schüler, ergab sich folgendes Bild: sechs boten das Schul-CommSy an, sechs lonet2, fünf gaben andere, nicht weiter spezifizierte an. Erfreulich ist hier ganz klar, dass alle Teilnehmer solche webbasierte Unterstützungen anbieten, die natürlich (wenn auch in leicht anderer Form) Einsatz in der Softwareentwicklung finden.

Aus der offenen Frage, ob Gruppenarbeit stattfinde und wenn ja, wie diese gestaltet sei, ergibt sich, dass Gruppenarbeit von nahezu allen Lehrern umgesetzt wurde (im Schnitt mit einer Gruppenstärke von zwei bis drei Teilnehmern), es jedoch häufig an Methoden mangelt, um diese zur Zufriedenheit der Lehrer und der Schüler durchzuführen. Es wurde geäußert, dass es gerade an der Aufgabenverteilung unter den Teilnehmern hapere, da es schwerfalle, dort das richtige Herangehen zu vermitteln. Darüber hinaus entsteht aus den Antworten der Eindruck, dass manche Gruppenarbeit trotzdem bedeutet, dass jeder Teilnehmer Aufgaben an einem eigenen PC bearbeitet. Dies zeigt insgesamt ganz klar, dass hier ein Verbesserungsbedarf besteht, der die sinnvolle Gruppenarbeit in geordnete Bahnen bringt. Auch hier liegt es nahe, sich der Projektmanagementmethoden der Softwareentwicklung zu bedienen und dies auch unter diesem Aspekt den Schülern zu vermitteln.

Gefragt, ob der Austausch von Daten zur Lösung von Aufgaben unter den Teilnehmern erlaubt ist und auf welchen Kanälen dies geschehe, gaben 13 der Befragten an, dass dieser erlaubt sei und in der Regel nicht in geregelten Bahnen verlaufe – meist per USB-Speicherstift. An dieser Stelle ist darauf hinzuweisen, dass solch ein Austausch nur mit geeigneten Werkzeugen und in geregelten Prozessen sinnvoll erscheint. Nur so wäre gewährleistet, dass alle Teilnehmer in gleichem Maße davon profitieren können. So könnte ein Gruppengefühl gestärkt werden, da die Lösung gemeinsam entwickelt und untereinander verbreitet würde.

Bei den Antworten zu der Frage, ob Dokumentationen angefertigt werden müssen und zu welchem Zeitpunkt dies geschehe, ist herauszulesen, dass diese eher zum Ende hin angefertigt werden müssen und diese Aufgabe von wenigen Schülern geschätzt

wird. Somit wird deutlich, dass attraktivere Herangehensweisen vorgestellt werden müssen, um Programmierergebnisse im Schulkontext bewerten zu können. Dazu gibt die Softwareentwicklung keine offensichtliche Antwort, da es dort nicht um Einzelleistungen, sondern um ein fertiges und funktionierendes Produkt geht. Trotzdem können Artefakte aus der Softwareentwicklung, die parallel zur Programmierung entstehen, zumindest in Teilen zur Dokumentation und Präsentation verwendet werden (siehe Kapitel 8.2.3).

Zusammengefasst scheint es gerade für die oben beschriebenen sozialen Aspekte und den Realitätsbezug an einfachen Methoden zu mangeln, die Lehrern mit auf den Weg gegeben werden können, um Gruppenarbeit zu vereinfachen und strukturiert verlaufen zu lassen. Darüber hinaus erweist es sich als wichtig, dass der spannende und kreative Entwicklungsprozess im Vordergrund steht und störende Nebenaufgaben, wie Abschlussdokumentationen, interessanter gestaltet werden bzw. entsprechend der Softwareentwicklung während des eigentliches Projekts parallel entstehen. Wie bereits erwähnt, bieten agile Methoden Praktiken an, um darauf reagieren zu können.

## 3.3 Kapitelzusammenfassung

Zunächst wurde herausgearbeitet, dass agile Methoden in der Lehre prinzipiell als sinnvoll und hilfreich angesehen werden, da sie zum einen die Lernergebnisse verbessern, aber auch eine positive Lernumgebung und Berufsaussicht vermitteln, um Informatik attraktiv wahrzunehmen. Problematisch dabei ist jedoch, dass sich diese Betrachtungen hauptsächlich auf die Hochschulbildung beziehen und somit deutlich zu spät ansetzen, um gestaltend in Bezug auf das Image der Informatik unter Jugendlichen mitwirken zu können.

An Schulen oder ähnlichen Einrichtungen kann nach Literaturlage und aufgrund einer Stichprobe argumentiert werden, dass dort der Eindruck entstehen kann, dass Programmieren eine Tätigkeit für Solisten ist, die Lösungen aus dem Nichts entwickeln und keinerlei soziale Interaktion dafür benötigen. Die vorhandenen Prozesse, die in Gruppenarbeit ablaufen, scheinen zumeist unstrukturiert und ohne Realitätsbezug, so dass verpasst wird, den Schülern die realen und sozialen Aspekte der Softwareentwicklung nahe zu bringen.

Somit bleibt festzustellen, dass bereits in der schulischen Bildung die Divergenz zu entdecken ist, die in Kapitel 2 bereits beschrieben wurde: Entgegen der Realität der Softwareentwicklung wird die Informatik in der Schule zu leicht als eine

Disziplin wahrgenommen, die keinen Wert auf soziale Interaktion legt. Erst an den Hochschulen wird dann versucht, diese Divergenz mittels *pair programming* oder ganzheitlicher Umsetzung der agilen Methoden wieder aufzuheben.

Einen weiteren Kontakt Jugendlicher zu digitalen Medien stellt die CCI dar und es gilt auch dort zu untersuchen, welchen Stellenwert zwischenmenschliche Interaktion hat. Einen wichtigen Sonderfall der CCI stellen die so genannten ILEs dar, da sie ausdrücklich als Lernumgebungen zum Programmieren konzipiert sind. Zunächst wird im kommenden Kapitel die CCI allgemein untersucht, woraufhin in Kapitel 5 dann ILEs vorgestellt werden.

# Kapitel 4

# Kinder und Jugendliche als Nutzergruppe

Dieses Kapitel stellt das Feld der Entwicklung von technischen Systemen/Umgebungen für und mit Kindern und Jugendlichen vor. Im Wesentlichen gibt das Kapitel einen Überblick, welche Sichtweisen in diesem Forschungsfeld vorherrschen und welche aktuell relevant sind. Darüber hinaus stellt dieses Kapitel die Frage, ob Kinder und Jugendliche lediglich als Einzelnutzer gesehen werden bzw. welche Kooperationsformen für diese Nutzergruppe existieren. Mögliche Bezüge zu realen Vorgehen der modernen Softwareentwicklung werden dargestellt.

Insgesamt stellt das in diesem Kapitel vorgestellte Forschungsfeld den Kontext dar, in dem der entwickelte Prototyp aus Kapitel 8.1 entstanden ist.

## 4.1 Gegenstand der Forschung

Im folgenden Abschnitt wird ein Überblick über ein recht junges Forschungsfeld angeboten, das sich mit der Entwicklung technischer Systeme für und mit Kindern und Jugendlichen beschäftigt. Dabei wird dieses Forschungsfeld zunächst zeitlich und inhaltlich eingeordnet. Darauf folgt eine Betrachtung des viel beachteten Ansatzes des Konstruktionismus, der als eine Grundlage heutiger Betrachtungen in dieser Disziplin anzusehen ist. Im abschließenden Teil wird darauf eingegangen, welche Sichtweisen, Erkenntnisse und Vorgehen sich inzwischen in der Wissenschaftsgemeinde durchgesetzt haben und die versprechen, eine wohlgestaltete Kind-Computer-Kommunikation zu entwerfen.

### 4.1.1 Geschichtliche Betrachtungen

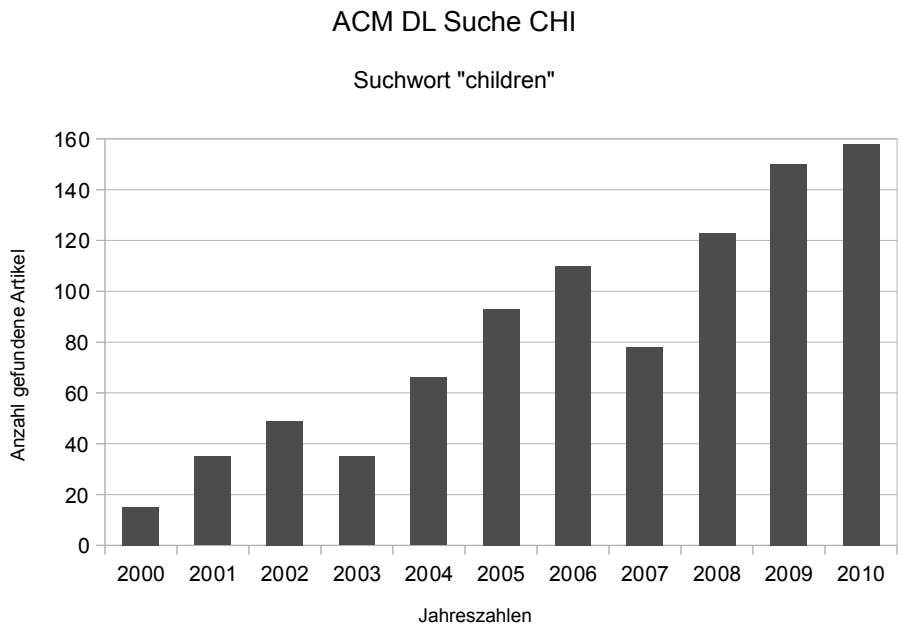



Abbildung 4.1: Anzahl gefundener Artikel zu CCI Themen bei der CHI über die letzten zehn Jahre (nach [Read et al., 2011]).

Aus der Informatik oder genauer der Mensch-Computer-Interaktion, der Psychologie und der Pädagogik entstand in den letzten Jahren ein Forschungsfeld, das sich damit befasst, wie Systemoberflächen und Anwendungen gestaltet werden müssen, damit Kinder und Jugendliche diese intuitiv verstehen und verwenden können. Zudem stellt sich immer die Frage, ob es spezielle Umgebungen gibt, die Kinder in ihrer Entwicklung fördern und unterstützen. Es gab von Papert [1980], Turkle [1984] zwar schon frühe Arbeiten, die sich der Thematik Kinder und Jugendliche im Dialog mit dem Computer annahmen, jedoch kann behauptet werden, dass erst seit Beginn der 2000er Jahre in angemessener Breite Forschungsvorhaben durchgeführt werden. Gut ersichtlich ist dies, da seit 2002 eine jährliche internationale Konferenz *Interaction Design and Children* (IDC) existiert, die ein hohes Ansehen und einen großen Zulauf genießt. Aus diesem wissenschaftlichen Umfeld ist inzwischen auch eine *IFIP TC13 Special Interest Group*[1] mit gleichem Namen entstanden. Darüber hinaus drängen diese Themen verstärkt auch in die klassische Mensch-Computer-Interaktion. So existiert beispielsweise ein angenommener Antrag für eine *Featured*

[1] http://www.idc-sig.org, zuletzt besucht am 15. April 2012.

*Community*[2] auf der *Conference on Human Factors in Computing Systems* (CHI) 2011, in dem dargelegt wird, dass das Themenfeld der *Child Computer Interaction* (CCI) seit 2000 stetig zunimmt. Erkenn- und messbar ist dies an Workshops, Konferenzen und Zeitschriften, die speziell auf dieses Themengebiet gerichtet sind, aber auch an der Anzahl der Publikationen in Proceeding der renommierten Konferenzen der Mensch-Computer-Interaktion, wie beispielsweise der CHI, der Nordichi und der Interact [Read et al., 2011]. Eine von Read et al. vorgeschlagene Übersicht über die Anzahl der veröffentlichten Artikel, die das Suchwort *children* enthalten und auf der CHI der letzten zehn Jahre vorgestellt wurden, verdeutlicht diesen Zuwachs deutlich (siehe Abbildung 4.1). Es liegt in der Natur der Sache, dass Read et al. die CCI als einen entscheidenden Faktor für die zukünftige Gestaltung und Weiterentwicklung der Mensch-Computer-Interaktion identifizieren.

### Konstruktionistische Ideen in der Informatik

Es existieren zwei Werke, die schon zu einem frühen Zeitpunkt das Verhältnis von Kindern und Jugendlichen zu Computerumgebungen beschreiben. In dem Buch von Seymour Papert [1980] wird davon berichtet, wie Schüler mit der Programmiersprache LOGO (siehe Kapitel 5.1) den Computer *programmieren* und so die Kontrolle über die Technologie der Gegenwart erlangen. Papert sieht in Computern bzw. Entwicklungsumgebungen wie LOGO die einmalige Gelegenheit, jedem Schüler ein Werkzeug an die Hand zu geben, mittels dessen sich dieser die Lerninhalte nach eigenen Herangehensweisen und mit eigenen Strukturen über das Konstruieren verschiedener Artefakte aneignen kann. In diesem Zusammenhang erkennt Papert deutlich eine praktische Erweiterung des Konstruktivismus nach Jean Piaget[3]. Papert nennt diese Erweiterung Konstruktionismus und ist der Überzeugung, dass Computer eine ideale Umgebung dafür darstellen. Er sieht es als grundlegend dafür an, dass Programmierumgebungen so zu gestalten sind, dass die jeweilige Programmiersprache so leicht zu erlernen ist wie eine Fremdsprache, die man in dem entsprechenden Land vor Ort lernt. Darüber hinaus sieht er im Computer die Möglichkeit, von Fehlern zu lernen, diese hinzunehmen, zu analysieren und Verbesserungen bzw. Berichtigungen darauf aufbauend zu entwickeln. Nur wenn diese Punkte berücksichtigt werden, handelt es sich um ein Werkzeug, das es dem jugendlichen Nutzer erlaubt, es nach

[2] http://chi2011.org/, zuletzt besucht am 15. April 2012.

[3] Piaget war ein Schweizer Entwicklungspsychologe, der zeigen konnte, dass Lernen besonders erfolgreich ist, wenn der Lernende einen Bezug zu den eigenen konstruktiven Prozessen herstellen kann.

seinen eigenen Denkstrukturen zu verwenden. Er hebt auch hervor, dass Computer nicht nur eine eigene Kultur hervorbringen, sondern auch im Dienste anderer Kulturen stehen können. Auf diese Weise können Jugendliche bei der Generierung eigener intellektueller Strukturen unterstützt werden, da es Computer erlauben, sich der umgebenden Kulturen zu bedienen. Für die Anwendung des Konstruktionismus in zukünftigen technologischen Umgebungen schwebt ihm die Umsetzung in der Art von Strukturen der brasilianischen Sambaschulen vor, in denen alle Teilnehmer (Anfänger, Profis, Kinder und Erwachsene) gemeinsam trainieren und sich ohne Hierarchien gegenseitig Hilfestellung und Feedback geben.

Untersucht man das Buch von Papert [1980] auf Hinweise nach konkretem kooperativen Arbeiten am Computer oder auch sozialen Aspekten in den beschriebenen Prozessen, so bleibt festzustellen, dass in den meisten Situationen[4] von dem einzelnen Kind vor einem jeweiligen Computer berichtet wird. Trotzdem lassen sich im Text immer wieder Hinweise finden, die darauf aufmerksam machen, dass soziale Interaktion auch in der Arbeit mit Computern wichtig ist. So wird z.B. im Buch von *Piagetian learning* gesprochen, um zu beschreiben, dass natürliches, spontanes Lernen immer dann auftritt, wenn Personen in Interaktion mit ihrer Umwelt, also auch anderen Personen, stehen. Aber auch die oben erwähnten Metaphern, die Papert bemüht, zeugen von einer sozialorientierten Ansicht: Die Ideale, Fremdsprachen im entsprechenden Land zu erlernen, vorhandene Kulturen zu verdeutlichen und die Strukturen einer Sambaschule anzunehmen, sind allesamt von einer hohen sozialen Interaktion und Kooperation geprägt. Darüber hinaus ist es Papert wichtig festzustellen, dass der Lehrer im Unterricht auch immer ein Lernender und somit Teil einer Gruppe ist. Statt durchzuführen, was der Lehrer sagt, sollen die Schüler tun, was der Lehrer tut. Dies veranschaulicht zugleich, dass Papert von einer Gruppe ausgeht, die gemeinsam Lösungen erarbeitet, wobei sich jeder Teilnehmer dieses Wissen in seiner eigenen Herangehensweise aneignen darf bzw. soll. Da durch das Programmieren Prozeduren verständlicher gemacht werden, sieht er auch die Chance, Wissen in mundgerechte bzw. nachvollziehbare Stücke (mind-size bits) zu zerlegen, um besser innerhalb einer Gruppe darüber kommunizieren zu können. Gegen Ende des Buchs beschreibt der Autor ungezwungene soziale Umgebungen (unstructured social situations) als Ideal. Hier kann nach seinen Worten die Interaktion zwischen allen Beteiligten gefördert und damit verhindert werden, dass Informationsfluss als Einbahnstraße verstanden wird. Papert geht davon aus, dass jeder Teilnehmer einer

[4]Die Ausnahme stellt ein Abschnitt dar, in dem die Rede von zwei Schülern ist, die gemeinsam einen Rechner verwenden [Papert, 1980, S. 78 ff.].

Lerngruppe den Meinungsaustausch anstrebt, um Fehler zu verstehen und beseitigen zu können, aber auch um Spaß an der Sache zu haben.

Ein zweites wichtiges bzw. grundlegendes Buch wurde von Sherry Turkle [1984] verfasst. Darin beschreibt sie unter anderem, dass Kinder Computerumgebungen in psychologischen Begriffen verstehen. Für Kinder ist der Computer ein echtes Gegenüber [Turkle, 1984, S. 51 ff.]. Erst später beginnen sie zu unterscheiden zwischen mechanischem Denken und selbstständigem, originellem Denken [Turkle, 1984, S. 64 ff.]. Neben der Betrachtung der Beziehung von Kindern und Jugendlichen zu Computerumgebungen stellt sie einige Vorteile für Lernprozesse heraus: Computer ermöglichen einen leichten Zugang zum Schreiben [Turkle, 1984, S. 114], Computer schaffen an Schulen eine neue positive Arbeitsatmosphäre, in der die Rolle des Lehrers eher als helfender Teilnehmer einer Gruppe zu verstehen ist [Turkle, 1984, S. 119]. Sie spricht von einer Gemeinschaftsbildung, die dadurch entsteht, dass Kinder und Jugendliche Ideen an Computern schnell umsetzen und *weiterspinnen* können [Turkle, 1984, S. 122]. Turkle [1984, S. 140 ff.] berichtet auch, dass Mädchen häufig die kreativen Aspekte der Programmierung unmittelbar erkennen und schätzen. Sie gibt zu bedenken, dass bei einigen Kindern Computerumgebungen zu Isolierung und Vereinsamung führen können, da ein Konkurrenzgedanke entstehen kann, der dazu veranlasst, dass eigene Lösungen den anderen Teilnehmen nicht zugänglich gemacht werden [Turkle, 1984, S. 158 ff.].

Zusammenfassend kann man den beiden Werken also entnehmen, dass die Autoren bereits zur damaligen Zeit ein großes Potenzial für den Einsatz von Computern mit Kindern und Jugendlichen sahen, da Computerumgebungen es vermögen, viele verschiedene Wege zum Wissenserwerb anzubieten. Gleichzeitig weisen sie jedoch – wenn auch implizit – darauf hin, dass soziale Komponenten verloren gehen können und daher aktiv im Umgang mit Computern in Lerngruppen gefördert werden sollten.

### Rollen im Design Prozess

Die vorgestellten Frühwerke veranschaulichen, dass Kinder und Jugendliche im Zusammenhang mit Computern einen wichtigen Forschungsgegenstand darstellen. Die grundlegende Idee der Forschungsgemeinschaft baut heutzutage darauf auf, dass bereits Kinder und Jugendliche in die Entwicklungsprozesse involviert sein sollten. Die zentralen Forschungsfragen zielen darauf ab, Lösungen zu entwickeln, die auf die speziellen Wünsche und Bedürfnisse von Kindern und Jugendlichen eingehen und

die Anwendbarkeit von Formen der Beteiligung herzustellen. Der Begriff der *children designers* wurde durch das Buch von Idit Harel [1991] geprägt. Dort beschreibt sie, wie Schüler gemeinsam computergestützte Lernumgebungen für jüngere Schüler entwickeln, welche Fragen sie sich dabei stellen, wie sie Aufgaben lösen und was sie dabei lernen. Sie fand unter anderem heraus, dass bereits Viertklässler in einer computergestützten Arbeitsumgebung hochgradig motiviert und gleichzeitig vollkommen in der Lage sind, eigene Lernziele zu definieren und einzuhalten. Harel ist davon überzeugt, dass Kinder und Jugendliche durchaus fähig sind, am Design Prozess von computergestützten Lernumgebungen und folglich auch anderen Systemen vollwertig teilzunehmen.

Die Methoden, die in dem Buch von Harel [1991] beschrieben werden, sind nicht nach den Grundsätzen des partizipativen Design gestaltet, aber es ist eindeutig, dass es auf denselben Ideen aufbaut. Aus diesem Grund stellten auch Scaife et al. [1997] fest, dass die Frage essentiell ist, ob für oder mit Kindern und Jugendlichen entwickelt wird. Hiermit ebneten sie den Weg für Untersuchungen, wie partizipatives Design in diesem Kontext anzuwenden ist. Scaife et al. schlugen eine Mischform von User Centered Design und partizipativem Design vor, um den Anforderungen von Kindern und Jugendlichen gerecht zu werden. Sie äußerten jedoch Bedenken bezüglich der Betrachtung von Kindern und Jugendlichen als vollwertige Designpartner (wie beim partizipativen Design mit erwachsenen Teilnehmern üblich), da sie nicht in der Lage wären, eigene Lernziele zu definieren. Sie schlugen daher Kinder und Jugendliche als „Informanten“ vor, die Prototypen testen, Änderungswünsche und Ideen in Interviews verbalisieren und Designvorschläge in (Spiel-)Szenarien entwickeln sollen. Diese Ansicht teilt jedoch Druin [2002] einige Jahre später nicht mehr. Sie identifiziert vier Rollen, die Kinder und Jugendliche im Designprozess einnehmen können und die sich herauskristallisiert haben: Nutzer, Tester, Informant, Designpartner (siehe Abbildung 4.2). Während ein Nutzer dabei beobachtet wird, wie dieser ein kommerzielles bzw. finales Produkt bedient, wird der Tester bereits vor Beendigung eines Projekts gebeten, Prototypen auszuprobieren und diese zu kommentieren. Von einem Informanten dagegen wird erwartet, dass dieser zwischen den Rollen, Nutzer und Tester wechselt und so Informationen über Systeme erarbeitet und den Entwicklern mitteilt. Ein Designpartner wird als vollständiges Mitglied des Entwicklerteams gesehen und ist so in die Gestaltung und Entwicklung eines Produkts mit einbezogen. Druin erkennt diese Rollen als historisch gewachsen und hierarchisch strukturiert an. Sie sieht daher in jeder Rolle auch Elemente aus den darunter liegenden Rollen. Während sie einräumt, dass es keine universelle Rolle

gibt, die für alle Projekte passt, stellt sie heraus, dass die Rolle des Designpartners zu den Resultaten führt, die am besten die Neigungen, Ansprüche und Wünsche von Kindern und Jugendlichen erfüllen. So gibt es einige Veröffentlichungen, die von dem erfolgreichen Einsatz von Kindern und Jugendlichen als Designpartner berichten [Druin et al., 1997; Druin und Fast, 2002; Garzotto, 2008].

Jedoch gibt es auch Quellen [Laughnan, 2004; Roussou et al., 2007], die darauf hinweisen, dass Ergebnisse, die mit Designpartnern entstehen, sich nur schwer verallgemeinern bzw. auf andere Domänen übertragen lassen. Dies macht deutlich, dass diese Herangehensweise für jedes neu zu entwickelnde Produkt einen hohen Aufwand voraussetzt und sowohl für viele Institute aber auch für Schulen aus zeitlichen und finanziellen Gründen wegen begrenzter Mitarbeiterzahlen nicht zu bewältigen ist.

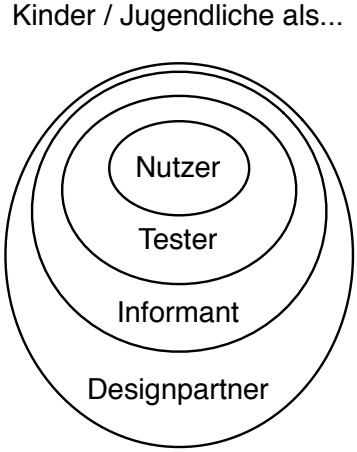



Abbildung 4.2: Die Rollen von Kindern und Jugendlichen im Designprozess (nach [Druin, 2002]).

Einen anderen Kritikpunkt heben Moraveji et al. [2007] hervor, indem sie darauf hinweisen, dass bei langjährigen Projekten mit Kindern und Jugendlichen als Designpartnern immer auch eine Auswahl der *passenden* Teilnehmer stattfindet. Somit ist davon auszugehen, dass die Ergebnisse immer nur auf eine spezielle Zielgruppe zugeschnitten sind, da anzunehmen ist, dass bestimmte Charaktere und Verhaltensweisen unter Kindern und Jugendlichen sich nicht mit dem Ideal des Designpartners vereinen lassen und deren Interessen sich somit auch nicht in den Ergebnissen widerspiegeln. Moraveji et al. fordern daher eine Durchführung von etlichen kleinen, kurzlebigen Projekten mit verschiedenen jugendlichen Teilnehmern, um über die Gesamtheit eine repräsentative Gruppe zu erhalten, an der dann Produkte ausgerichtet werden können. Darüber hinaus heben sie hervor, dass es wichtig ist, einfache Methoden zu entwickeln und anzubieten, um auch anderen Wissenschaftlern oder Lehrern die Möglichkeit zu bieten, darauf zurückzugreifen. Moraveji et al. stellen in

diesem Zusammenhang *Comicboarding* vor, das angelehnt an *Storyboarding* aus dem partizipativen Design Jugendlichen ermöglichen soll, eigene Ideen zu verbalisieren. Dabei wird besonders darauf Wert gelegt, dass sich diese Technik auch für einen kurzzeitigen Kontakt zu Entwicklern eignet, bei dem aus verschiedenen Gründen keine Umgebung geschaffen werden kann, in der sich Jugendliche als Designpartner etablieren. Grundidee des *Comicboarding* ist, dass ein Comiczeichner auf Zuruf der jugendlichen Teilnehmer Szenarien in Comicpanels überführt. Moraveji et al. beschreiben *Comicboarding* in mehreren Aufbauten (Comiczeichner vor Ort/an anderem Ort, Sprechblasen ausfüllen, Geschichte zwischen Anfangs- und Endbild entwickeln, ganze Geschichte entwerfen) und vergleichen diese mit dem herkömmlichen *Storyboarding*. Die Variante, die am meisten Ideen von Jugendlichen hervorbrachte, war diejenige, die jeweils ein Anfangs- und ein Endbild anbot und der Comiczeichner vor Ort war. Moraveji et al. geben zu Bedenken, dass die Comicästhetik nicht jedem Thema gerecht werden kann und dass es zu Abhängigkeiten bezüglich des Comiczeichners kommen kann. Nichts desto trotz sehen sie in *Comicboarding* eine vielversprechende Möglichkeit, um in kleinen Projekten mit Kindern und Jugendlichen gemeinsam Ideen zu generieren.

### 4.1.2 Einordnung aktueller Sichtweisen

Inzwischen scheint es – der Diskussion der CCI und den in diesem Umfeld publizierten Veröffentlichungen zufolge – erwartet zu werden, dass Kinder und Jugendliche als vollwertige Designpartner angesehen werden. Es gibt zahlreiche Beispiele dafür, dass dieser Ansatz zu guten Ergebnissen führt; es wird dabei jedoch oft vergessen, dass es sich dabei meist um mehrjährige und generös geförderte interdisziplinäre Forschungsprojekte handelt, die leider nur in einer geringen Anzahl stattfinden. Es ist davon auszugehen, dass es gerade kleinen Projekten immer wieder an anwendbaren kostengünstigen und kleinformatigen Methoden mangelt und so viele relevante Ideen zur Verbesserung der CCI nicht zu erfolgreichen Ergebnissen gebracht werden. Die Forschungsgemeinschaft scheint darauf noch nicht reagiert zu haben, um z.B. auch Lehrern oder einzelnen Forschern zu ermöglichen, Systeme zu entwickeln, die von Kindern leicht und gerne angenommen werden und darüber hinaus ein positives Bild der IT zeichnen. Hier ist möglicherweise auch die Aussage von Papert [1980,

S.32 f.] hilfreich, bei der er das QWERTY-Phänomen[5] als Warnung begreift, dass Strömungen und Annahmen gerade im Bereich Computer und Lernen ständig überprüft werden müssen, um zu verhindern, dass man in Sackgassen gerät, für die darin der Aufwand zu groß ist, sie wieder zu verlassen. Eine mögliche Sackgasse könnte also sein, nur Erkenntnissen aus umfassenden, langjährigen Studien zu folgen und dementsprechend nur auf aufwändige Methoden zu setzen, die einen großen und häufig nicht realisierbaren (Zeit-)Aufwand für Schüler sowie Lehrer bzw. Wissenschaftler darstellen. Viele kleinere Ideen und Ansätze würden so nicht mehr aufgegriffen. Die vorliegende Arbeit versteht sich folglich als ein Beitrag, der auf vielerlei verschiedene und kurze Projekte gemäß der Aktionsforschung setzt. Dies geschieht in der Hoffnung, Ergebnisse präsentieren zu können, die aufgrund ihrer leichtgewichtigen Verfahren als praxistauglich für die Informatikbildung im Schulkontext angesehen werden können.

## 4.2 Gemeinsames Lernen in Computerumgebungen

Des Weiteren gilt es zu untersuchen, wie Kinder und Jugendliche als Nutzer von technischen Systemen in sozialen Handlungsweisen unterstützt werden, da ja bereits Papert [1980] und Turkle [1984] vor möglichen Isolationen der Nutzer warnten. Relativ wenig Hinweise finden sich dazu im originären Forschungsfeld der CCI, so dass hier zunächst allgemeine Erkenntnisse zum Lernen am Computer in Gruppen betrachtet werden. O'Malley [1992] beschreibt beispielsweise solche Lernumgebungen für verschiedene informatikfremde Disziplinen, die durch Computer unterstützt werden sollen. Sie kommt dabei zu dem Schluss, dass computergestützte Lernsysteme soziale Komponenten aufweisen müssen. Für sie muss solch eine Lernumgebung sowohl synchrone als auch asynchrone Kommunikation zwischen allen Beteiligten erlauben. O'Malley sieht die gemeinsame Nutzung eines Rechners als kommunikatives Werkzeug, da die Repräsentationen darin mit mindestens einem Partner diskutiert werden können. Dazu führt sie eine Studie an, in der sie bei den Teilnehmern feststellen konnte, dass ihnen gerade für Diskussionen auf Meta-Ebene (z.B. Vorgehensplanung) Augenkontakt sehr wichtig war. Dieser Kontakt wurde zumeist non-verbal hergestellt, so dass sie davon ausgeht, dass Lernumgebungen zumindest in Teilen

[5]Es handelt sich hierbei, um die Durchsetzung der QWERTY-Tastaturbelegung als Standard, wobei diese Standardisierung nicht in einer besonders guten Nutzbarkeit zu begründen ist. Vielmehr wurde diese Belegung gewählt, um bei Schreibmaschinen zu verhindern, dass die Typenhebel sich gegenseitig behindern. Trotz zahlreicher Studien, die in Folge für den Menschen bessere Belegungen vorschlugen, setzte sich QWERTY durch, da sie bereits zu weit verbreitet war.

die Teilnehmer am selben Ort zur selben Zeit vereinen. O'Malley berichtet auch von der Notwendigkeit einer Lernumgebung, um von den Erfahrung im Klassenraum auf die reale Situation in der entsprechenden Domäne schließen zu können bzw. Verknüpfungen methodisch-didaktisch herzustellen.

Es bleibt also festzuhalten, dass in anderen Lerndomänen, die auf Computerunterstützung zählen, darauf Wert gelegt wird, dass diese Umgebungen soziale Interaktion anbieten und fördern. In diesem Sinne kann man hier die Grundlagen des *blended learning* erkennen, wonach technische Systeme niemals als alleinige Lern- und Lehrplattform begriffen werden sollten.

### 4.2.1 Lehr- und Lernplattformen

Um den von O'Malley [1992] genannten Punkten bezüglich der zusätzlichen technischen Unterstützung von sozial geprägten Lernprozesse zu entsprechen, entstanden Onlineplattformen für Lehre und Lernen, die zunächst generische und fachunspezifische Räume für Projekte bzw. Gruppen darstellen und erst durch die Teilnehmer inhaltlich einer Fachdomäne angepasst werden. In diesen abgeschlossenen Räumen können Informationen, Ergebnisse und Artefakte der einzelnen Teilnehmer ausgetauscht und besprochen werden. Die Interaktion zwischen den Teilnehmern wird auf verschiedenen Ebenen angeboten, so dass beispielsweise allgemein zugängliche Diskussionen gestartet werden können, aber auch Privatnachrichten versandt werden können. Darüber hinaus können meistens Aufgaben verteilt werden und Termine festgelegt werden. Gängige entsprechende Plattformen sind das CommSy[6], moodle[7] und lo-net2[8].

#### CommSy

Das CommSy stellt eine Lehr- und Lernplattform zur Schaffung von kooperativen Wissensnetzen dar, die am Fachbereich Informatik der Universität Hamburg entwickelt wurde [Jackewitz et al., 2004]. Hierbei handelt es sich um eine webbasierte Kooperationsoftware, mit der Personengruppen exklusive Räume für Projekte erstellen können, die dann auch nur den registrierten Teilnehmern zugänglich sind. Das CommSy bietet in den Projekträumen folgende Funktionen an:

[6] http://www.commsy.net/, zuletzt besucht am 15. April 2012.
[7] http://moodle.org/, zuletzt besucht am 15. April 2012.
[8] http://www.lo-net2.de/, zuletzt besucht am 15. April 2012.

- Ankündigungen veröffentlichen und kommentieren
- Aufgaben erstellen und verteilen
- Diskussionen initiieren und dazu beitragen
- (Unter-)Gruppen bilden zur weiteren Interaktion
- Materialien bereitstellen und kommentieren
- Personen direkt kontaktieren
- Termine vereinbaren

Dem CommSy liegt zugrunde, dass sämtliche Inhalte mit semantischem Kontext abgespeichert werden und wieder durch Verschlagwortlichung, Suchfunktion oder Kategorisierung gefunden und selektiert werden. Dabei setzt das CommSy zu großen Teilen auf kooperative Prozesse, da Schlagworte und Kategorisierungen erst durch deren Verwendung durch mehrere Nutzer das volle Potenzial entfalten [Finck et al., 2005]. Somit existiert dort nicht die Ordnermetapher, die z.B. bei Dateisystemen zu finden ist. Otto [2002] berichtet von einem 21 Unterrichtsstunden umfassenden Einsatz des CommSys im Projektunterricht in der neunten Klasse eines Gymnasiums. Projektinhalt war das Thema Lebensmittel – eingebettet in den Chemie- und Erdkundeunterricht. Ausdrücklich weist Otto darauf hin, dass die Ergebnisse zur Projektarbeit mit dem CommSy fach- und schulformunabhängig sind. Das CommSy unterstützte die Projektteilnehmer in der gemeinsamen Planung, der „handlungsorientierten Auseinandersetzung“ mit der Thematik und der Ergebnispräsentation. In der Möglichkeit zu direktem Feedback und der guten Nachvollziehbarkeit der Gruppenprozesse sieht Otto Gründe, die für den weiteren Einsatz des CommSys im Schulunterricht sprechen. Er lobt die leichte Erlernbarkeit des CommSys und berichtet, dass einige Schüler den CommSy-Raum auch nach Beendigung des Projekts weiter nutzten, um Daten für andere Fächer auszutauschen. Da alle Medien, Aussagen und Kommentare allen zugänglich sind, wird eine Infrastruktur bereitgestellt, die es Schülern ermöglicht, sich untereinander zu helfen. Er vermutet, dass Schüler es nicht gewohnt sind, unfertige Ergebnisse anderen zur Verfügung zu stellen, da erst zum Ende des Projekts der CommSy-Raum mit Ergebnissen befüllt wurde. Auch die prinzipielle Möglichkeit, in der Freizeit von zuhause etwas zum Projekt beizutragen, wurde nur selten genutzt.

Das CommSy wird weitläufig an der gesamten Universität Hamburg zur Unterstützung der Lehre verwendet[9], und es existiert eine Infrastruktur, die auch für Forschungszwecke zur Verfügung steht. Das CommSy wird des Weiteren durch die

[9]https://www.commsy.uni-hamburg.de/, zuletzt besucht am 15. April 2012.

Behörde für Schule und Berufsbildung Hamburg und das Hamburger Landesinstitut Lehrerbildung und Schulentwicklung allen Hamburger Schulen zugänglich gemacht und ist somit auch an vielen Hamburger Schulen in Verwendung[10].

### 4.2.2 CCI Plattformen

Hier wird nun betrachtet, welche Stellenwerte gemeinsames Lernen und kooperative Prozesse in der CCI haben. In Kapitel 5 wird dann unter anderem darauf eingegangen, welche Formen der Kooperation in den am weitesten verbreiteten Entwicklungsumgebungen für Kinder und Jugendliche bzw. Programmieranfänger angeboten werden.

Wie bereits beschrieben, ist die CCI eine breit gefächerte Disziplin, es gibt zahlreiche Systemprototypen für verschiedenste Einsatzgebiete und Aufgaben mit unterschiedlichsten Randbedingungen, so dass es nicht sinnvoll erscheint, eine allgemeine Aussage über Kooperationsformen zu treffen. Es empfiehlt sich vielmehr, zentrale Themen zu identifizieren und diese zu bewerten. Eine wiederkehrende Thematik der CCI ist das *digital storytelling*, hierzu gibt es zahlreiche Publikationen, so dass ausgeschlossen werden kann, dass es sich um eine kurzlebige Erscheinung handelt. Darüber hinaus eignet es sich als Betrachtungsgegenstand, da gerade im *traditionellen storytelling* (also ohne Computerunterstützung) viele Kooperationsformen existieren und angesprochen werden und es häufig das Ziel des *digital storytelling* ist, sinnvolle technische Erweiterungen anzubieten, die jedoch die ursprünglichen Aktivitäten und Vorteile beibehalten. Diese Untergruppe der CCI wird daher nachfolgend beispielhaft dahingehend untersucht, welche Kooperationsformen zu finden sind, wie darauf eingegangen wird und welche Vorteile darin gesehen werden. Schließlich kann an diesem Beispiel gut verdeutlicht werden, dass hier noch elementare Untersuchungen anstehen und auf zukünftige klare Aussagen zu warten ist, die sich den Kooperationsformen von Kindern und Jugendlichen an Computerumgebungen widmen.

Eine Untersuchung aller veröffentlichten Papiere (insgesamt 13 *full papers*) zu *digital storytelling* mit Kindern und Jugendlichen bei namhaften Konferenzen (IDC, CHI und *Computer Supported Cooperative Learning*) ergab, dass dort in erster Linie Wert darauf gelegt wird, jungen Nutzern technische Umgebungen zugänglich zu machen und ihnen die Entwicklung einer multimedialen Geschichte zu ermöglichen [Göttel, 2011c]. Zwar finden sich häufig in den einleitenden Worten der Artikel Bezüge zu kooperativen Zielen, jedoch zumeist nur derart, dass die Teilnehmer in

[10] http://hamburg.schulcommsy.de/, zuletzt besucht am 15. April 2012.

| | Entfernter Ort | Selber Ort | Angereicherte soziale Umgebung |
|---|---|---|---|
| Gestalten | | | |
| Austauschen | | | |
| Darstellen | | | |

Tabelle 4.1: Aspekte des *digital storytelling* und der Kooperationsformen. Die angereicherte soziale Umgebung beschreibt Systeme, die soziale Interaktionen, z.B. gemeinsames Spiel mit realen Gegenständen, in eine Computerumgebung integrieren.

Gruppen an jeweils eigenen Computerumgebungen arbeiten und hinterher die fertigen Geschichten präsentieren. Nur wenige Arbeiten legen darauf Wert, dass der ursprüngliche Gedanke des storytelling auch im *digital storytelling* beibehalten oder gar erweitert wird, lediglich Stanton et al. [2001] beschreiben diesen Vorgang explizit. Mit der Systemumgebung von Stanton et al. ist es möglich, Geschichten gemeinsam zu erstellen, untereinander Ideen und Konzepte auszutauschen, um dann wiederum gemeinsam die finalen Geschichten darzustellen oder durchzuspielen. Alle Aktionen werden auch von Lehrern und Betreuern in einen Kontext gesetzt, in dem typische realweltliche Gruppendynamiken von Klassenverbänden und Körperaktivitäten berücksichtigt werden. Die Untersuchung ergab darüber hinaus, dass je drei Aspekte des *digital storytellings* und drei Kooperationsformen den betrachteten Systemen zugrunde liegen. Die daraus erstellte Matrix wird in Tabelle 4.1 gezeigt. Um im Verlauf des gesamten Arbeitsprozesses Kooperation anzuleiten, soll ein System pro Zeile der Matrix mindestens eine passende Funktionalität anbieten. Wünschenswert wäre jedoch eine höhere Abdeckung der Ausprägungen in der Matrix. Dies gelingt Stanton et al., deren Arbeit somit sicherlich auch als Vorbild für Lernumgebungen zu sehen ist, die das Programmieren an sich vermitteln sollen. In der Tat finden sich hier auch Gemeinsamkeiten mit den zugrunde liegenden Konzepten aktueller ILEs. Sie werden daher in Kapitel 5 diesbezüglich vorgestellt und untersucht. Eine detaillierte Betrachtung der einzelnen Aspekte und die Einordnung der insgesamt 13 relevanten Artikel über *digital storytelling* ist im Rahmen dieser Arbeit nicht zielführend, kann aber bei Göttel [2011c] nachgelesen werden.

Betrachtet man die oben beschriebenen Strömungen der heutigen CCI, so wird deutlich, dass der Aspekt des gemeinsamen Arbeitens oder Lernens immer mehr in den Hintergrund getreten ist oder oberflächlich beachtet wird. Dies ist verwunderlich, da eine weitere frühzeitige und häufig zitierte Arbeit existiert, die eigentlich schon deutlich auf die Vorteile der Gruppenunterstützung von jugendlichen Pro-

grammierern hingewiesen hat [Bruckman, 1998]. Bruckman stellt darin ein textbasiertes *Multi User Dungeon* (MUD)[11] mit dem Namen *Moose Crossing* vor und beschreibt damit die Vorteile des gemeinsamen Lernens für den Konstruktionismus nach Papert [1980]. *Moose Crossing* erlaubt es Nutzern, eigene Welten und Objekte mit einer proprietären Programmiersprache zu erstellen und diese dann inhaltlich mit selbst verfassten Texten zu beschreiben. Nutzer in *Moose Crossing* treffen andere Teilnehmer in den virtuellen Räumen und können mit diesen über einen Chat in Kontakt treten. Erfahrene Nutzer waren zum einen häufig damit beschäftigt, neue Teilnehmer zu empfangen und ihnen die ersten Schritte zum Programmieren attraktiv zu vermitteln. Zum anderen hat sich herausgebildet, dass es enorm motivierend und wichtig war, eigene Kreationen anderen zu zeigen. Hier hat sich eine freundschaftliche Atmosphäre gebildet, die für fremde Kreationen Respekt und konstruktive Vorschläge zur Erweiterung hervorbrachte. Bruckman benennt drei positive Faktoren für diese sozialorientierte Lernumgebung: Die Teilnehmer von *Moose Crossing* lernen aus eigener Motivation, selbstgesteuert und werden von der Interaktion mit einer gleichberechtigten Nutzer-Gemeinschaft gestützt und gestärkt durch die Möglichkeit, Hilfe zu bekommen, aber auch Hilfen anbieten zu können. Des Weiteren geht Bruckman davon aus, dass Treffen der Teilnehmer in der realen Welt, die für den Projektzeitraum organisiert wurden, dazu beigetragen haben, eine Gemeinschaft zu formen, die eine positive und konstruktive Lernatmosphäre fördert. Bruckman stellt heraus, dass *Moose Crossing* folgende Eigenschaften bereithält, die den Teilnehmern besonders helfen:

- Es existieren positive Vorbilder (unabhängig von gesellschaftlichen Hierarchien, Geschlecht oder Alter), die die Fähigkeit zu programmieren besitzen und bereit sind, das Wissen zu teilen.

- Die Projekte der Teilnehmer laden zum Mitmachen ein, erscheinen dynamisch und allgegenwärtig.

- Soziale und emotionale Unterstützung wird gewährleistet, um Technikängste überwinden zu können.

- Durch die Gemeinschaft oder die Möglichkeit, *Experten* per E-Mail zu kontaktieren (wenn auch wenig genutzt), ist meist technische Unterstützung abrufbar.

[11] MUDs waren in den 70er und 80er Jahren sehr beliebt. Vereinfacht gesagt handelt es sich dabei um textbasierte *Adventures*, die zeitgleich mit weiteren Nutzern über das Internet gemeinsam erlebt werden können.

- Alle Teilnehmer stellen eine Gemeinschaft dar, die an der Arbeit anderer interessiert ist und viel Lob ausspricht; so entsteht die Möglichkeit, die eigene Arbeit angemessen zu präsentieren und gewürdigt zu sehen.

Nach Bruckman ist für das Auftreten solcher positiven Eigenschaften entscheidend, dass es sich bei *Moose Crossing* zumeist um Hilfe von menschlichen Teilnehmern handelt und nicht um computergenerierte. Für Bruckman ist aber auch klar, dass diese sozialen Eigenschaften konkret auf Systemeigenschaften zurückzuführen sind: *Moose Crossing* bietet technisch die Möglichkeit, andere zu kontaktieren, zum Mitmachen einzuladen und auf Ergebnisse oder abgeschlossene Projekte hinzuweisen. Sie benennt daher diese drei Punkte als Grundelemente für technische Umgebungen, um damit konstruktionistisches Lernen fördern zu können. Sie sieht diese technischen Elemente – ähnlich zum *blended learning* – im Zusammenspiel mit zwischenmenschlicher Interaktion.

Es ist aus den genannten Gründen verwunderlich, dass diese Erkenntnisse und Systemeigenschaften später kaum bzw. nur teilweise (vergleiche dazu Kapitel 5.2) aufgegriffen wurden. Ein Begründungsversuch besteht darin, dass MUDs kaum noch eine Rolle in der aktuellen Anwendungslandschaft der CCI bzw. in der gesellschaftlichen Wahrnehmung spielen. Die von Bruckman genannten Punkte eigenen sich im Rahmen dieser Arbeit besonders, um sozialorientierte Aspekte von ILEs zu prüfen und werden daher in Kapitel 9.1 zu diesem Zweck verwendet. Dieses Verfahren stellt auch eine Besinnung der vorliegenden Arbeit auf die von Bruckman genannten Punkte sicher, da diese gut die Grundsätze von Papert [1980] und die Ideale der CCI allgemein in einen Bezug zu sozialen Aspekten setzen. Bruckman [1998] spricht von einer *community of learners*, die soziale Interaktion hervorbringt und somit positive Assoziationen bei den Lernenden hervorruft und den konstruktionistischen Grundgedanken entspricht. Naheliegend ist es daher zu fordern, dass genau solche *communities of learners* in der Informatikbildung entstehen, um gute Lernerfolge zu erzielen und um auf sozialorientierte Aspekte der Informatik aufmerksam zu machen und diese so attraktiver präsentieren zu können. Es ist erklärtes Ziel der vorliegenden Arbeit, Schritte in diese Richtung einzuleiten, sowohl aus technischer Sicht als auch mit angepassten Lehrmethoden für den Informatikunterricht (vergleiche Kapitel 8).

Die Arbeit von Henriksen et al. [2010] ist, wie bereits in Kapitel 1.2.1 angeführt, eine der wenigen aktuellen Arbeiten, die soziale Aspekte wieder aufgreift. Henriksen et al. haben ebenfalls ausgemacht, dass Jugendliche soziale Interaktion und Prozesse nicht in Verbindung mit Informatik oder IT bringen. Auch Henriksen et al. sehen es demnach als essentiell an, bereits vor der universitären Lehre den Fokus

auf sozialorientierte Aspekte der Softwareentwicklung zu legen. Sie stellen dafür eine Online *community* namens *Greenfoot Gallery* (siehe Abbildung 4.3) vor, die das ILE *Greenfoot* (Kapitel 5.2.2) erweitert. Nutzer von Greenfoot haben hier die Möglichkeit, eigene Projekte hochzuladen und so anderen Teilnehmern zugänglich zu machen. Die Projekte lassen sich innerhalb eines beliebigen Webbrowsers nutzen. Alle angemeldeten Teilnehmer können Projekte kommentieren, bewerten, in Foren diskutieren und Galerien (*Collections*) – bestehend aus eigenen oder favorisierten Projekten – anlegen. Die Startseite der *Greenfoot Gallery* zeigt stets eine Vielzahl an empfohlenen neuen Projekten und besonders beliebten Projekten. In einer sechswöchigen Untersuchung der *Greenfoot Gallery*[12] stellten Henriksen et al. fest, dass Kommentare, bei einem Durchschnittswert von 5,2 Kommentaren je Projekt, durchweg positiver Art waren. Selbst unfertige oder fehlerhafte Projekte erhielten aufmunternde oder konstruktive Kommentare. In den Kommentaren erkannten Henriksen et al. fünf Grundtypen: Ermutigend (44%), Technische Diskussion (37%), Ideenaustausch (14%), Ressourcen-Austausch (3%) und sonstige Kommentare (5%). Es wurden auch Kommentare gefunden, die in mehrere dieser Grundstrukturen fielen, daher übersteigen die addierten Prozentzahlen die 100%. Sie fassen die Vorteile solch einer Plattform wie folgt zusammen: Durch die Möglichkeit, Projekte anderen zugänglich zu machen, wird die Motivation gesteigert. Die aufmunternden Kommentare vereinfachen es, diese Motivation länger aufrecht zu erhalten. Der gemeinsame rege Austausch von Ressourcen ermöglicht eine Verbesserung der Ergebnisse, da sie den Gedanken der professionellen Softwareentwicklung entsprechend auf vorhandenen Lösungen aufbauen. Bei der Beschreibung von *Greenfoot* in Kapitel 5.2.2 wird noch einmal die *Greenfoot Gallery* aufgegriffen und mit einem ähnlichen Angebot verglichen, welches für *scratch* existiert, zu dem jedoch momentan keine gesonderte Veröffentlichung zur Verfügung steht. Kritisch angemerkt werden muss jedoch, dass beide Plattformen einige Punkte nach O'Malley [1992] vernachlässigen. So erlauben beispielsweise diese Plattformen in ihrer Beschreibung nur schwer eine gemeinsame Nutzung eines Rechners für synchrone Kommunikation, die auf Augenkontakt beruht.

[12] Zeitraum 10.12.2008–21.01.2009; Insgesamt wurden 70 Projekte in dieser Zeit der *Greenfoot Gallery* neu oder in überarbeiteter Version hinzugefügt.


Abbildung 4.3: Die Startseite der *Greenfoot Gallery*. Hier werden neue Szenarien, Kommentare und Diskussionsbeiträge als Stream visualisiert.

## 4.3 Anleihen aus der Softwareentwicklung

Sucht man nach Hinweisen auf den Zusammenhang von Computerlernumgebungen und professionellen Praktiken, so wird man bei Papert [1980] fündig. In seinen Ausführungen beschreibt er nötige mehrfache Iterationen, um eine zufriedenstellende Lösung zu entwickeln. In diesem Sinne erwähnt er dort bereits Anleihen aus der Berufswelt [Papert, 1980, S. 30]:

> „The image of children using the computer as a writing instrument is a particularly good example of my general thesis that what is good for professionals is good for children.“

Damit kann man argumentieren, dass es in Lernumgebungen wünschenswert ist, Kindern und Jugendlichen Praktiken aus der Berufswelt zu vermitteln. Vergleicht man in diesem Zusammenhang sozialorientierte Aspekte in der CCI mit der heutigen professionellen Softwareentwicklung, so erkennt man Potenzial zur Verbesserung durch die bereits genannten, sozialorientierten Praktiken der agilen Methoden, die in Kapitel 2.2 (Seite 15 f.) allgemein beschrieben wurden und in Kapitel 3.1.1 (Seite 29 f.) im Bildungskontext betrachtet wurden. Angepasste und in acht Projekten erprobte Praktiken der agilen Methoden werden dann noch einmal in Kapitel 8.2 (Seite 147 f.) vorgestellt.

## 4.4 Kapitelzusammenfassung

In diesem Kapitel wurde ein Überblick über das relativ junge Forschungsfeld CCI gegeben. Betrachtet wurden dabei grundlegende Arbeiten von Papert [1980] und Turkle [1984], die als erste Kinder und Jugendliche im Umgang mit Computerumgebungen beobachteten. In einer historischen Betrachtung wurden die Strömungen der *community* beschrieben, die zur heutigen Ansicht geführt haben: Kinder und Jugendliche werden idealerweise als vollwertige Designpartner empfohlen, um so auf die speziellen Bedürfnisse dieser eingehen zu können und angepasste Lösungen zu entwickeln. Dies erfordert meist einen großen Aufwand an Zeit, Gruppengröße, beteiligten Institutionen und Geldmitteln. Vereinzelt existieren Stimmen, die daher empfehlen in kleineren kurz angelegten Projekten in verschiedenen Kontexten mit Jugendlichen zu arbeiten und daraus leichtgewichtige Praktiken zu entwickeln. Dies stellt eine aussagekräftige Sichtweise dar. Beispielhaft wurde das *Comicboarding*, eine adaptierte Technik aus dem partizipativen Design vorgestellt, die es ermöglicht, auch in kleiner angelegten Projekten, Kinder und Jugendliche als Informanten zu sehen.

Eine weitere Betrachtung legt offen, dass die Frage, in welcher Form Kinder und Jugendliche an Computern zusammenarbeiten sollen, in der CCI Literatur bisher nicht adäquat thematisiert bzw. untersucht wurde. Es gibt zwar allgemeine Forderungen an Lehr- und Lernumgebungen, die von allgemein gehaltenen Onlineplattformen abgedeckt werden, in der CCI jedoch nur von einigen wenigen Systemen um-

gesetzt werden. Dies wurde am Beispiel der Arbeiten zum *digital storytelling* deutlich. Die wenigen Arbeiten, die einen klaren Bezug zu sozialorientierten Aspekten beim Programmieren herstellen, beispielsweise *Moose Crossing* oder *Greenfoot Gallery*, stellen positive Ausnahmen dar, auch wenn insbesondere die *Greenfoot Gallery* nicht sämtlichen, allgemeinen Forderungen für soziale Lehr- und Lernumgebungen entspricht.

Des Weiteren wurde in diesem Kapitel darauf hingewiesen, dass Bezüge zu professionellen Praktiken in Lernumgebungen durchaus sinnvoll sind. Dies würde es beispielsweise nahe legen, die heutige Softwareentwicklung, sprich agile Methoden und CGK-Erkenntnisse, in Lernumgebungen zu integrieren.

Das nachfolgende Kapitel beschreibt ILEs als Unterkategorie der CCI und stellt detailliert *scratch* und *Greenfoot* vor und untersucht den Aspekt, ob Bezüge zur realen Softwareentwicklung für Jugendliche ableitbar sind.

# Kapitel 5

# Entwicklungsumgebungen für Kinder und Jugendliche

Schon seit der Entwicklung der Programmiersprachen *COBOL* und *BASIC* Anfang der 1960er Jahre existieren Bemühungen, Programmierumgebungen zu erschaffen, die einen einfachen Einstieg in das Programmieren und die darunter liegenden Programmierkonzepte ermöglichen. Zahlreiche entsprechende Systeme sind seitdem entstanden. Abschnitt 5.1 dieses Kapitels stellt historische Meilensteile der Entwicklungsumgebungen oder -sprachen für Jugendliche bzw. Programmieranfänger vor.

Des Weiteren werden Grundanforderungen an computergestützte Lernumgebungen anhand historischer Systeme untersucht. Dies zeigt, dass diese selten ohne soziale Komponenten auskommen.

In Abschnitt 5.2 werden die sogenannten *initial learning environments* (ILE) vorgestellt, die zur Zeit sowohl im schulischen als auch im freizeitlichen Kontext weit verbreitet sind und auch nach wissenschaftlichen Maßstäben als besonders gelungen anzusehen sind. Es handelt sich hierbei um *scratch* und *Greenfoot*.

## 5.1 Geschichtliche Betrachtungen

Schon frühzeitig befassten sich Wissenschaftler mit der Problematik, ob Programmiersprachen für Novizen zu hohe Einstiegshürden darstellen (z.B. [Knuth und Merner, 1961]). So lässt sich sagen, dass bereits ab 1960[1] Programmiersprachen und Programmierumgebungen entwickelt wurden, die speziell auf die Bedürfnisse von An-

[1] Also z.B. noch vor der Entwicklung des ersten Computerspiels *Spacewar!*, das 1962 am *Massachusetts Institute of Technology* vorgestellt wurde oder der Erfindung der Computer Maus, die 1968 von Douglas Engelbart der Öffentlichkeit präsentiert wurde.

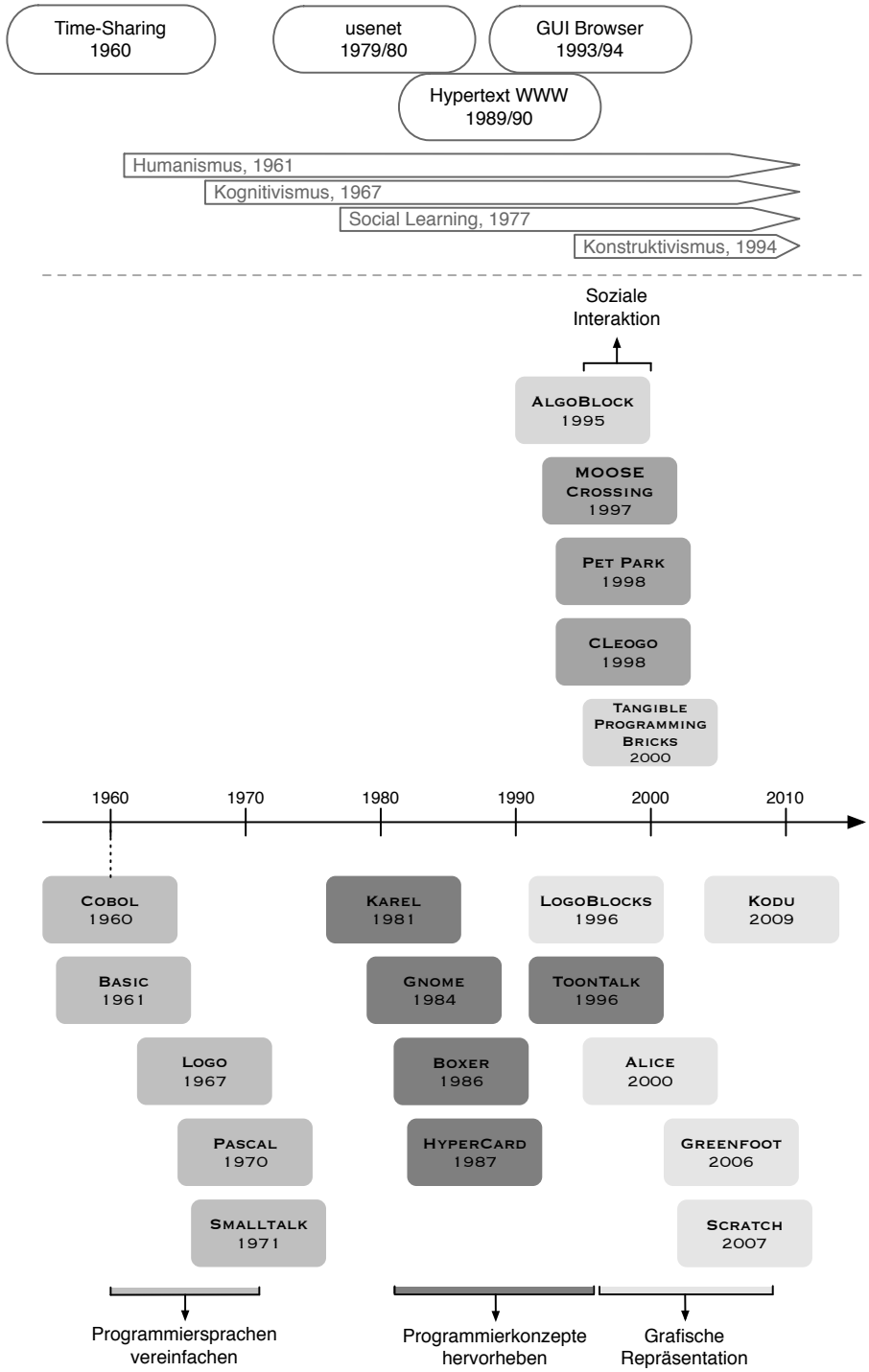



Abbildung 5.1: Zeitleiste für Entwicklungsumgebungen im Bildungskontext. Zeitliche Angaben stammen von Kelleher und Pausch [2005] (Entwicklungsumgebungen), Ashworth et al. [2004] (Lerntheorien) und Hellige [2008] (Internettechnologien). Die Angaben zu Lerntheorien und Internettechnologien dienen der Orientierung und werden im Text nicht weiter erörtert.

fängern eingehen. Hierzu gibt es eine hilfreiche Taxonomie von Kelleher und Pausch [2005], die diese Unterfangen in verschiedene Aspekte und Lernziele unterteilen. Zur anschaulichen Betrachtung im Kontext dieses Kapitels wurden einige Systeme in eine Zeitachse übertragen. Die in der Abbildung 5.1 vorgestellten Umgebungen geben bei Weitem nicht die vollständige Breite der von Kelleher und Pausch betrachteten Umgebungen an. Zur Wahrung der Übersicht stellt die Abbildung 5.1 die Systeme vor, die von Kelleher und Pausch als einflussreich (anhand von wissenschaftlichen Referenzen auf diese Sprachen/Umgebungen) identifiziert wurden. Darüber hinaus sind der Abbildung Umgebungen hinzugefügt, die für die vorliegende Arbeit besondere Relevanz haben (gekennzeichnet als *soziale Interaktion*) und teilweise erst nach der Veröffentlichung von Kelleher und Pausch erschienen sind. Es geht ausdrücklich nicht darum, einen vollständigen geschichtlichen Überblick zu geben, es fehlen z.B. auch Systeme, die sich großer Beliebtheit unter Lehrern erfreuen, beispielsweise *Squeak EToys*[2]. Es kann jedoch davon ausgegangen werden, dass die in der Zeitleiste genannten Systeme alle wichtigen Aspekte abdecken, die auch in anderen nicht genannten Systemen als Grundgedanken zu erkennen sind. Ziel dieser Darstellung ist es, bestimmte Strömungen, Wechselwirkungen und Ideologien über die Jahre zu identifizieren, um daraus Rückschlüsse für diese Arbeit ziehen zu können.

Die Programmiersprachen bzw. Programmierumgebungen, die Abbildung 5.1 zeigt, lassen sich nach vier verschiedenen zugrunde liegenden Intentionen einteilen:

- *Programmiersprachen vereinfachen*: Hier ist die Idee zu erkennen, dass die Sprache an sich vereinfacht werden soll, z.B. über eine anschaulichere Syntax, die der gesprochenen Sprache näher kommt.

- *Programmierkonzepte hervorheben*: Der Grundgedanke hierbei ist, dass Konzepte der Programmierung besser vermittelt werden sollen, wie z.B. der Veranschaulichung von Befehlsfolgen, die man einem Roboter vorgibt und von diesem schrittweise und anschaulich abgearbeitet werden.

- *Soziale Interaktion*: Diese Systeme sind von dem Verständnis geprägt, dass Lernen einen sozialen Prozess darstellt, der von Systemen umfassend unterstützt werden sollte.

[2] *Squeak Etoys* ist eine grafische Programmierumgebung, die in *Squeak* – eine vollwertige objektorientierte Programmiersprache inklusive Entwicklungsumgebung – integriert ist und somit die Möglichkeit bietet, die erzielten Ergebnisse auch auf Textebene weiter zu verwenden bzw. zu bearbeiten.

- *Grafische Repräsentation*: Diese Systeme stehen dafür, dass dem Nutzer möglichst früh grafische Repräsentationen angeboten werden sollten, sei es in Form von Bausteinen, mit denen man Programme zusammenstellt, oder in Form von Szenarien, in denen man Methoden von Objekten per Hand am Objekt selbst aufrufen kann.

Im Folgenden wird zu den ersten beiden Intentionen jeweils ein System präsentiert. Zur sozialen Interaktion werden alle vier Umgebungen vorgestellt, da sie für die vorliegende Arbeit als grundlegend angesehen werden können. Nachdem bereits in Kapitel 4.2 allgemeine Anforderungen an Computerlernumgebungen benannt wurden, stellt Kapitel 5.2 *scratch* und *Greenfoot* als Vertreter der letzten Intention vor. Diese sind aktuell an Schulen und in Freizeiteinrichtungen für Jugendliche im weiten Einsatz und stellen somit den aktuellen Stand der Wissenschaft dar. Die beiden ILEs greifen auch einige Teilaspekte der *sozialen Interaktion* auf. In der vorliegenden Arbeit wird dementsprechend die Auffassung vertreten, dass weitere erfolgreiche Punkte aus der sozialen Interaktion in diese beiden ILEs einfließen müssen, um die Informatik in ein besseres Licht zu rücken.

### 5.1.1 Programmiersprachen vereinfachen

#### Logo

Bei Logo handelt es sich um einen Dialekt von *Lisp*, der so entwickelt wurde, dass Kinder und Jugendliche in der Syntax keine zu große Hürde sehen. Prinzipiell handelt es sich um eine vollwertige Programmiersprache, die auch dazu gedacht war, z.B. den Mathematikunterricht an Schulen zu unterstützen. Am bekanntesten ist Logo jedoch für seine *Logo Turtle*, der man einfache Anweisungen geben kann, die sie dann befolgt. Es handelt sich dabei um einen Akteur, der sich in einem definierten Feld schrittweise bewegen kann und einen Punkt am aktuellen Ort zeichnet (sofern diese Funktion aktiviert wurde), so dass Bewegungen in eine Richtung Linien entstehen lässt. Mit Anweisungen wie `forward 10` können Nutzer dem Akteur befehlen, zehn Schritte in Blickrichtung des Akteurs vorwärts zu gehen. Über Anweisungsfolgen ist es so möglich, Bilder zu zeichnen [Kelleher und Pausch, 2005, S. 113].

### 5.1.2 Programmierkonzepte hervorheben

#### Karel

Karel kann man als eine reduzierte Programmiersprache verstehen, die zum Einsatz in Einführungsveranstaltungen vor dem Einsatz echter Programmiersprachen gedacht ist. Es handelt sich um einen virtuellen Roboter, dem man in einer einfachen zweidimensionalen Welt mit Räumen, Straßen und Gegenständen Anweisungen geben kann. Im Vergleich zur *Logo Turtle* ist eine bewusste Reduktion der Programmiersprache auf ausschließlich dieses Szenario gewählt worden, um Anfängern die Komplexität einer herkömmlichen Programmiersprache zu ersparen und erst einmal die Abläufe und Ideen des Programmierens zu vermitteln. Es ist Nutzern möglich, einzelne Schritte einer Abfolge zu betrachten und auszuführen. Darüber hinaus können Prozeduren definiert werden, die eine Abfolge von Anweisungen darstellen und in Folge als eigene Anweisung verwendet werden können [Kelleher und Pausch, 2005, S. 103].

### 5.1.3 Soziale Interaktion

#### AlgoBlock

Bei AlgoBlock handelt es sich um eine Umgebung, die für Teilnehmer von Lerngruppen am selben Ort und zur selben Zeit konzipiert ist. Der Aufbau von AlgoBlock beinhaltet mehrere Bausteine, die jeweils für bestimmte grundlegende Logo-Befehle stehen, und einen Rechner. Die Bausteine sind so gestaltet, dass sie an einem Tisch gemeinsam mit anderen Teilnehmern verwendet und zusammengestellt werden können. Die Zusammenstellung der Bausteine hat immer eine Auswirkung auf die Programmumgebung, die am Monitor des Rechners präsentiert wird. Kinder und Jugendliche verlieren durch den Bausteincharakter die Scheu vor dem Programmieren und lernen dabei, dass sie gemeinsam Konzepte hinterfragen und erforschen können. Mit AlgoBlock sind jedoch nur recht einfache Programmieranweisungen zu realisieren [Kelleher und Pausch, 2005, S. 106].

#### Moose Crossing

*Moose Crossing* wurde bereits in Kapitel 4.2.2 (S. 63 ff.) ausführlich vorgestellt. Es handelt sich um ein System, das es Nutzern über das Internet erlaubt, gemeinsam Orte oder Artefakte zu programmieren, diese zu besuchen oder sich gegenseitig

vorzustellen. Auch über das eigentliche System hinaus ist es möglich, Kontakt untereinander aufzunehmen (organisierte Teilnehmertreffen/Kurse, E-Mail-Verteiler oder Foren) [Bruckman, 1998; Kelleher und Pausch, 2005].

### Pet Park

Pet Park versucht die Gedanken von *Moose Crossing* in eine zweidimensionale Darstellungsebene zu transferieren. Dem Nutzer stehen mehrere Hunde zur Auswahl, die mit verschiedenen Animationen ausgestattet sind. Diese Animationen lassen sich über Skripte oder virtuelle Bausteine verbinden und so dem eigenen Hund zuweisen, um selbst definierte Kombinationen als Aktionen ausführen zu lassen. Wie in *Moose Crossing* handelt es sich um eine virtuelle Welt für mehrere Benutzer, in der man gemeinsam diese Programmierungen vornehmen und eigene Kreationen vorstellen kann. Darüber hinaus ist es auch hier möglich, Räume zu erschaffen, die von anderen Nutzern besucht werden können. Dabei handelt es sich jedoch eher um einen Editor, der es erlaubt, Einrichtungsgegenstände zu wählen und im Raum anzuordnen [Kelleher und Pausch, 2005, S. 107].

### Cleogo

Bei Cleogo handelt es sich um eine Netzwerkumsetzung von Leogo, die es Nutzern erlaubt, an einer gemeinsamen Leogo Arbeitsumgebung zu programmieren. Leogo ist eine Weiterentwicklung des *Logo Turtle* Konzepts, in dem drei verschiedene Möglichkeiten angeboten werden, um dem Akteur Anweisungen zu geben: Textanweisungen, direkte Manipulation und Vorlagenbausteine [Kelleher und Pausch, 2005, S. 98]. Ziel von Cleogo ist es, gemeinsames Editieren und Betrachten eines Dokuments zu ermöglichen. Es existieren jedoch keinerlei technische Funktionen zur Kommunikationsunterstützung. Somit wird davon ausgegangen, dass sich entweder die Teilnehmer im selben Raum befinden oder andere zusätzliche Kommunikationskanäle verwenden [Kelleher und Pausch, 2005, S. 107].

### Tangible Programming Bricks

Wie bei AlgoBlock ist Tangible Programming Bricks ein System, das voraussetzt, dass die Teilnehmer sich zur selben Zeit am selben Ort befinden. Es handelt sich hierbei sogar um eine ähnliche Metapher: Legobausteine repräsentieren Programmbefehle, die dann zusammengesteckt werden können. Darüber hinaus erlauben spezielle Mikrochips, Bausteinen bestimmte Befehle zuzuweisen. Auch hier program-

mieren Kinder gemeinsam mithilfe von realen Gegenständen, um Einstiegshürden zu verringern und gemeinsame Aktivitäten zu fördern [Kelleher und Pausch, 2005, S. 106].

#### Gemeinsamkeiten

Zusammenfassend lässt sich sagen, dass die vorgestellten Systeme Wert darauf legen, dass Kinder und Jugendliche dazu animiert werden, über die Konzepte zu diskutieren und Berührungsängste zu verlieren, sei es durch die Verwendung von Bausteinen oder die Gewissheit, dass andere menschliche Personen erreichbar sind, die bereits programmieren können und gewillt sind, ihr Wissen zu teilen. Am besten scheint dies zu gelingen, wenn eine Gemeinschaft und rege menschliche Interaktion um das System und die jeweiligen Projekte aufgebaut werden.

## 5.2 Aktuelle Entwicklungsumgebungen

Folgend werden die ILEs beschrieben, die im Zuge dieser Arbeit in Projekten mit Schülern (siehe Kapitel 6) verwendet wurden und nach Kapitel 5.1 in ihrer Intention der *grafischen Repräsentation* zuzuordnen sind. Die Wahl fiel auf *scratch* und *Greenfoot*, da beide im Einsatz an deutschsprachigen Schulen sind und über eine wachsende *community* verfügen, die sowohl aus jugendlichen Nutzern als auch Lehrern und Dozenten besteht. Beide Umgebungen sind darüber hinaus weiterhin in der Entwicklung und Gegenstand von Forschungsgruppen[3]. Da außer Frage stand, dass sich *scratch* besonders für den Anfangsunterricht (siehe Kapitel 6.1) eignet, wurden die meisten der in Kapitel 6 ausführlich beschriebenen Projekte mit *scratch* durchgeführt. Zur Überprüfung der Beobachtungen der *scratch*-Projekte wurde ein Projekt mit *Greenfoot* durchgeführt.

### 5.2.1 Scratch

In gewisser Weise greift *scratch* die Ideen der Programmierumgebungen *AlgoBlock* und *Tangible Programming Bricks* aus Kapitel 5.1.3 auf, indem es eine Vielzahl von Bausteinen anbietet, die zusammengesteckt Anweisungsfolgen darstellen können. Der deutliche Unterschied zu den beiden genannten Umgebungen liegt darin, dass es sich um eine grafische Programmierumgebung handelt und somit sämtliche

[3]Für *scratch* ist die *Lifelong Kindergarten Group* des *Massachusetts Institute of Technology* federführend und die *Computing Education Group* der *University of Kent* bei *Greenfoot*.

Aktionen virtuell in der Computerumgebung durchgeführt werden. Daher werden keine realen, greifbaren Bausteine benötigt. Diese klare Referenz zu den historischen Programmierumgebungen, die auf soziale Interaktion Wert legen, ist zu einem späteren Zeitpunkt bei der Analyse der sozialorientierten Aspekten von *scratch* noch einmal von Belang (siehe Kapitel 9.3.1).

Die folgende Beschreibung von *scratch* stützt sich auf die praktische Anwendung und den Artikel von Maloney et al. [2010]. Da zum Zeitpunkt der Implementation des in Kapitel 8.1 vorgestellten Prototyps *scratch* nur in Version 1.3.1 zur Verfügung stand, beziehen sich alle Beschreibungen auf diese Version, obwohl *scratch* inzwischen in der Version 1.4 existiert. *Scratch* ist grundsätzlich für alle gängigen Betriebssysteme verfügbar (Windows, Mac OS X, Linux), wobei sich die Linux-Version ausdrücklich in einem beta-Stadium befindet und die Anwendbarkeit häufig von der Initiative der *scratch*-Nutzergemeinschaft abhängig ist. *Scratch* existiert in zahlreichen Übersetzungen, darunter auch Deutsch.

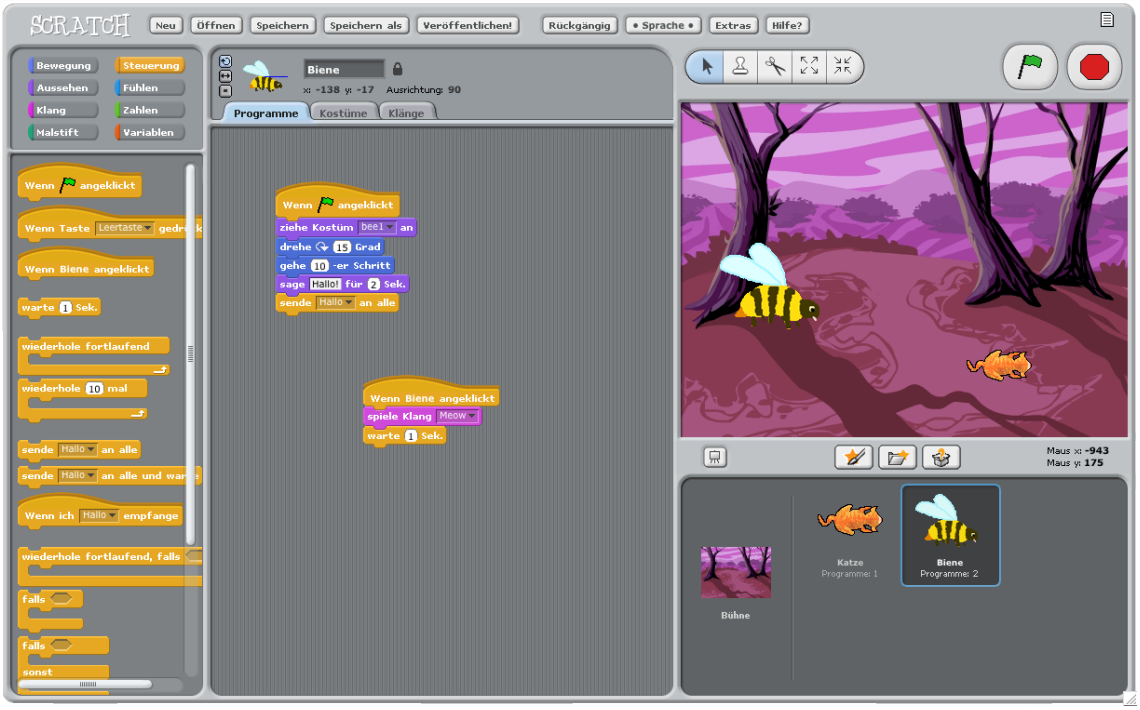



Abbildung 5.2: Die Oberfläche von *scratch* 1.3.1 mit den vier Abschnitten Bausteinbibliothek, Skripteditor, Bühne und Objektübersicht.

Die Oberfläche von *scratch* bietet immer nur ein Programmfenster an, um den Nutzern zu jeder Zeit sämtliche Möglichkeiten zur Verfügung zu stellen. Dieses Programmfenster ist in vier Abschnitte eingeteilt (siehe Abbildung 5.2):

- *Bausteinbibliothek*: Auf der linken Seite befindet sich eine Palette mit Kommandos, die als Bausteine dargestellt und in verschiedenen Kategorien zusammengefasst werden.

- *Skripteditor*: Im mittleren Bereich werden die Bausteine mittels *drag and drop* zu Skripten zusammengeführt. Diese Skripte sind immer für das jeweils ausgewählte Sprite (Figur bzw. Objekt) gültig.

- *Bühne*: Der obere rechte Bereich präsentiert die Bühne, auf der sämtliche Aktionen der Sprites mit den jeweiligen Skripten dargestellt werden. Die Bühne lässt sich auch mittels eines Knopfs auf Präsentationsmodus stellen, um ausschließlich das Ergebnis präsentieren zu können.

- *Objektübersicht*: In der unteren rechten Ecke werden die vorhandenen Sprites und die Bühne als Miniatur präsentiert; diese dienen zur Auswahl, um ihnen dann Skripte zuweisen zu können.

Die Bausteinbibliothek bietet folgende Kategorien an: Bewegung, Aussehen, Klang, Malstift, Steuerung, Fühlen, Zahlen und Variablen. Ein Mausklick auf die jeweilige Kategorie präsentiert im darunter liegenden Bereich sämtliche Bausteine, die zur jeweiligen Kategorie gehören. Die Bausteine sind ebenso wie die Kategorien farblich codiert, so dass die Zugehörigkeit der Bausteine auch dann noch zu erkennen ist, wenn sie zur Verwendung in den mittleren Arbeitsbereich verschoben wurden. Die Bausteine sind sprechend bzw. funktionsbezogen benannt, so dass sich im Idealfall ihre Funktion daraus ablesen lässt. Es existieren Kommando-Bausteine, die für eine fest definierte Aktion stehen und deutlich mehr Bausteine, die dem Nutzer Anpassungsmöglichkeiten erlauben (z.B. ist bei dem Baustein „gehe 10-er Schritt“ die Zahl veränderbar, sowohl ganzzahlig als auch als Kommazahl). Die Bausteine haben verschiedene Formen, so dass sichergestellt werden kann, dass nur Skripte zusammengesteckt werden können, die syntaktisch Sinn ergeben. Die Kategorie „Steuerung“ bietet Bausteine an, die Programmabläufe festlegen. Daher werden hier einige Bausteine genannt, die häufig Verwendung finden, um lauffähige Programme mit *scratch* zu entwickeln, es handelt sich dabei um *trigger blocks* und *control structure command blocks* [Maloney et al., 2010, S. 8]. Der *trigger block* „Wenn Fahne angeklickt“ dient als Startpunkt zur Ausführung von Anweisungen, da das Anklicken der Fahne über der Bühne (siehe S. 5.2.1) als initialer Befehl an sämtliche vorhandenen Sprites und die Bühne geschickt wird und von diesen Objekten mit dem genannten Baustein abgefangen werden kann. In ähnlicher Weise existieren *trigger blocks* für

Tastatureingaben, Anklicken des Objekts oder das Empfangen von selbst zu definierenden Nachrichten. Mittels des „sende an alle“-Bausteins ist es möglich, selbst definierte Nachrichten zu erstellen und zu senden. Der „falls“-Baustein bietet die Möglichkeit, eine Aktion durchzuführen, sofern ein bestimmtes selbst definiertes Ereignis eintritt. Dafür ist hinter dem „falls“ ein Platzhalter, der durch seine Form nur *function blocks* zulässt, die einen Wahrheitswert liefern. In der Kategorie „Zahlen“ existieren Bausteine, die es auch erlauben, eigene syntaktisch kompatible Abfragen zu erstellen (z.B. mit dem „und“-Baustein).

Im mittleren Arbeitsbereich wird jeweils oben das aktuell ausgewählte Sprite dargestellt. Man hat dort auch die Möglichkeit, einen Namen für dieses Sprite zu vergeben und die Drehrichtungen von *frei drehbar* über *rechts/links drehbar* auf *nicht drehbar* festzulegen. Darunter werden drei Reiter angeboten: „Programme“, „Kostüme“ und „Klänge“. Das Hauptaugenmerk liegt sicherlich auf „Programme“, da dort die Skripte für das Sprite zusammengebaut werden. „Kostüme“ und „Klänge“ bieten die Option, dem Sprite ein anderes Aussehen oder bestimmte Klänge zuzuweisen, die dann wieder unter „Programme“ mit bestimmten Bausteinen verwendet bzw. angesprochen werden können. Während der Programmausführung ist es möglich, das jeweils aktive Skript an einer weißen Umrandung zu erkennen. Ein Rechtsklick mit der Maus im mittleren Arbeitsbereich bietet die Optionen an, den Arbeitsplatz aufzuräumen (alle Skripte werden vertikal ausgerichtet), das Programm (alle vorhanden Skripte des ausgewählten Sprites) als Bild abzusichern und eine Anmerkung hinzuzufügen (Kommentar-Block).

Der Bühnenbereich erlaubt, das Programm mit einem „Fahne“-Knopf zu starten (sofern mindestens ein Objekt darauf reagiert) oder mit einem roten Stoppschild alle Skripte und damit das Programm zu stoppen. Darüber hinaus stehen noch fünf Knöpfe zur Verfügung, von denen exklusiv immer nur eine Option auszuwählen ist und diese jeweils die Nutzung des Mauszeigers bzw. des Mausklicks ändert; so kann man neben der normalen Mausfunktion duplizieren, löschen und Objekte vergrößern oder verkleinern.

Der Bereich, der die Bühne und die vorhandenen Sprites als Miniatur anbietet, erlaubt es mittels Knöpfen neue Sprites hinzuzufügen. Hierbei stehen dem Nutzer die Möglichkeiten zur Verfügung, ein Sprite zu malen (es öffnet sich ein Malprogramm in einem neuen Fenster), ein existierendes Sprite auf dem Computersystem auszuwählen oder ein zufälliges vorgegeben zu bekommen. Links neben diesen Knöpfen findet man einen weiteren Knopf, der den Wechsel zum Präsentationsmodus ermöglicht und so semantisch eher zur Bühne gehört.

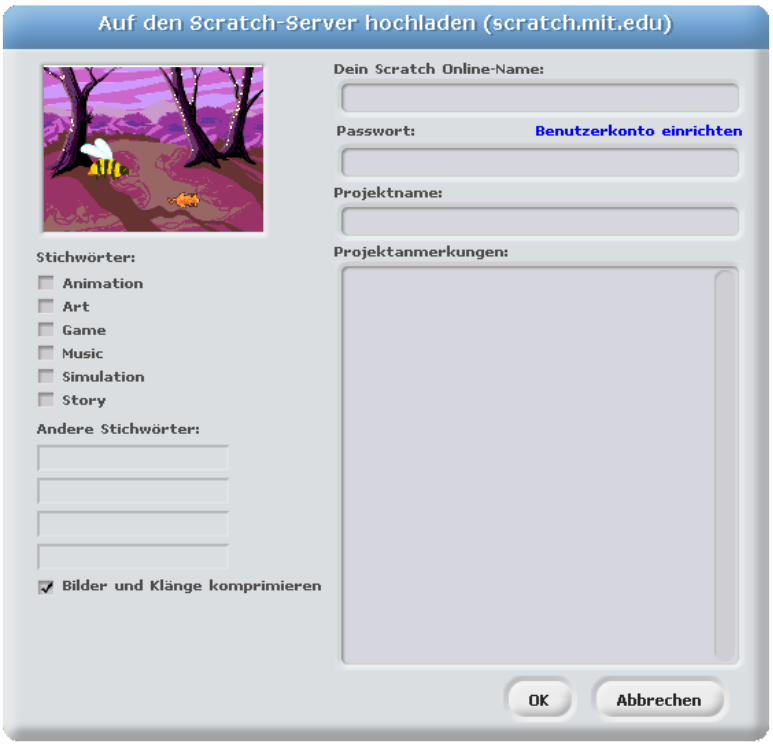



Abbildung 5.3: Das Dialogfenster zum Hochladen von *scratch*-Projekten unter *scratch* 1.3.1.

Über Knöpfe in einer übergeordneten Leiste können gängige Aktionen wie „Öffnen“, „Speichern“ und „Rückgängig“ ausgeführt werden. Es ist aber auch über den „Mitteilen!“-Knopf möglich, ein fertiges Projekt direkt aus *scratch* heraus auf die Website bzw. die Projektgalerie hochzuladen. Dazu öffnet sich ein Dialogfenster, das einen Nutzernamen und Passwort erfordert und dazu auffordert, das *scratch*-Projekt zu dokumentieren, zu verschlagwortlichen und zu benennen (siehe Abbildung 5.3). Der „Extras“-Knopf bietet an, Projekte zu importieren, die Ausführung des erstellten Programms in Einzelschritten zu starten und Klänge oder Bilder zu komprimieren.

Die Projektgalerie von *scratch* (siehe Abbildung 5.4) orientiert sich an bekannten und erfolgreichen Internet-Videoportalen wie *youtube* und *vimeo* und ist gleichzeitig die Startseite von *scratch*. Die gesamte Plattform kann in 44 verschiedenen Sprachen dargestellt werden[4]. Die Nutzer können hochgeladene *scratch*-Projekte direkt im Webbrowser abspielen. Auf der Startseite werden verschiedene Projekte vorgestellt bzw. empfohlen und verlinkt. Jedes hochgeladene *scratch*-Projekt hat eine eigene, eindeutige Einstiegsseite, die das jeweilige *scratch*-Projekt in einem Fenster darstellt und eine nebenstehende Textbeschreibung des Projekts. Sollte der Autor

[4]http://scratch.mit.edu/, zuletzt besucht am 15. April 2012.

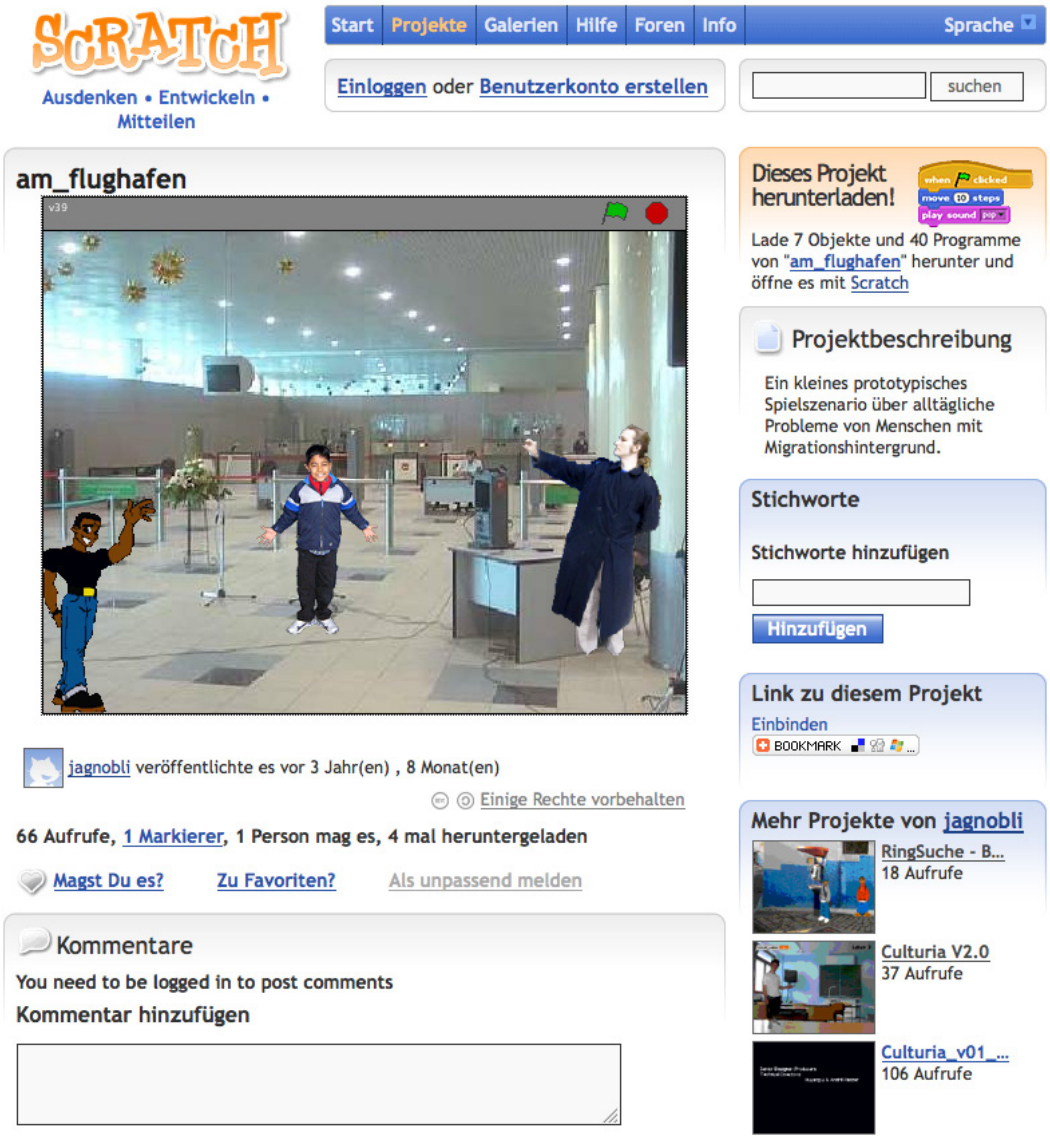



Abbildung 5.4: Eine *scratch*-Projektseite in deutscher Übersetzung.

mehrere Projekte hochgeladen haben, so sind auch diese verlinkt. Darüber hinaus können Projekten Schlagworte zugewiesen werden. Ein Verweis zu einem Projekt kann ebenfalls von dieser Seite aus über zahlreiche *social media* Dienste (z.B. E-Mail, *Twitter* oder *Facebook*) erstellt werden. Angemeldeten Nutzern ist es möglich, Kommentare zu *scratch*-Projekten zu hinterlassen und sich das ausführbare Projekt herunterzuladen. Danach kann es in *scratch* geöffnet werden, um dort die Skripte nachzuvollziehen oder bearbeiten zu können. Darüber hinaus kann jeder Nutzer Galerien erstellen, um *scratch*-Projekte in einen beliebigen Kontext zu setzen. In Foren können allgemeine oder spezifische Fragen gestellt bzw. beantwortet werden.

Folgend seien noch ein paar Aspekte genannt, die zur Einordnung von *scratch* im Verhältnis zu anderen Programmiersprachen hilfreich sind. Die Sprites können als Objekte wahrgenommen werden, da sie über ihre Variablen einen Zustand beschreiben und über ihre Skripte Methoden anbieten. Gleichwohl gibt es keine Klassen und somit auch keine Vererbung. Daher ist *scratch* nicht als objektorientierte Programmiersprache zu sehen. *Scratch* bietet außerdem keine Möglichkeit, um Prozeduren zu erstellen. Die Entwickler entschieden sich bewusst für ein schlankes Design von *scratch* und wollen in Beobachtungen an einer Vorversion von *scratch*, die Prozeduren ermöglichte, herausgefunden haben, dass Prozeduren zu verwirrend für die Zielgruppe sind [Maloney et al., 2010, S. 12]. Für eine noch detailliertere Betrachtung von *scratch*, besonders zur Betrachtung der vorhandenen Konzepte wie Variablen, Listen aber auch des *multi-threading* sei Maloney et al. [2010] empfohlen.

*Scratch* ist in Beziehung zur Matrix aus Kapitel 4.2.2 darauf ausgelegt, die Prozesse des Gestaltens bzw. Entwickelns Einzelner vor Ort und an entfernten Orten zu unterstützen.

### *Scratch* im deutschsprachigen Raum

*Scratch* ist aktuell in der deutschsprachigen Didaktik der Informatik viel diskutiert. Der folgende Überblick stellt diese Diskussion dar.

In einem Artikel von Romeike [2007a] wurden die vermeintlichen Vorzüge von *scratch* zur Einführung von Programmierinhalten und der kreativen Herangehensweise der Informatikbildung in Schulen (Gymnasium, Klassenstufe 11) beschrieben:

- Sowohl imperative als auch objektorientierte Konzepte lassen sich mit *scratch* vermitteln.
- Durch die intuitive Oberfläche von *scratch* können sich Schüler explorativ und in Eigenverantwortung davon überzeugen, dass Informatik Spaß machen kann.
- Eine einfache Präsentation der Ergebnisse ist durch die zugehörige Webplattform und die Möglichkeit gewährleistet, *scratch* auf allen gängigen Betriebssystemen zu installieren und so Freunde und Verwandte dazu anzuregen, eigene Programme zu entwickeln.
- *scratch* bietet eine motivierende Umgebung, die schnell zu ansehnlichen Ergebnissen führen kann, gleichzeitig jedoch auch das Erstellen von anspruchsvollen Programmen ermöglicht.

Auf die positiven Aspekte nach Romeike [2007a] aufbauend, berichten Stoll et al. [2008] von einem erfolgreichen Einsatz von *scratch* in der Klassenstufe 8 („Lernbereich 2: Informationen verarbeiten: Modell - Algorithmus - Lösung“). Unter anderem werden die Schüler in dem vorgeschlagenen Konzept mittels *scratch* an selbständige Problemlösungen in Einzel-, Partner- oder Gruppenarbeit herangeführt. Verglichen mit den Bildungsstandards der GI [Gesellschaft für Informatik e.V., 2008] bescheinigen Stoll et al. *scratch* tabellarisch eine relativ breite Abdeckung der dort geforderten Lernziele. Betrachtet man Aspekte der Tabelle, die soziale Anteile aufweisen („Begründen und Bewerten“, „Strukturieren und Vernetzen“, „Kommunizieren und Kooperieren“, „Darstellen und Interpretieren“), so fällt auf, dass die Abdeckung den Autoren zufolge zwar positiv ausfällt, sich jedoch keinerlei konkrete Bezüge zu technischen Gegebenheiten von *scratch* finden lassen.

Diesen beiden lobenden Artikeln widerspricht Baumann [2009a,b] gleich zweifach. Im Vergleich zu *TurtleArt*[5] spricht Baumann [2009b] *scratch* die Eignung für den Unterricht oberhalb der Grundschuljahrgänge ab. Er vermisst informatische Substanz in den meisten *scratch*-Projekten (sowohl in den vorgestellten Projekten von Stoll et al. [2008], als auch bei denen, die über die *scratch*-Webseite zu betrachten sind) und benennt die fehlende prozedurale Abstraktionsmöglichkeit bei *scratch* als großes Hindernis, da er Lernen im Zusammenhang mit Abstraktion sieht und diese Möglichkeit auch von Programmierumgebungen einfordert. Im Vergleich zu *Squeak Etoys* sieht Baumann [2009a] *scratch* ebenfalls im Nachteil. Diesmal bemängelt er zum einen, dass *scratch* mit vorgefertigten Bildeinteilungen daherkommt und so dem Lernenden die Möglichkeit der eigenen Anpassungen nimmt. Zum anderen führt er an, dass die recht schnellen Erfolgserlebnisse bei *scratch* auf lange Sicht negativ ausfallen würden, wohingegen *Squeak Etoys* durch die vergleichsweise komplizierte Handhabung „mehr Anreize für eigenes Gestalten“ bieten soll. Baumann [2009a] schließt mit der Behauptung, dass der Einsatz von *scratch* in der Oberstufe der „Missachtung der Fähigkeiten“ von Schülern gleichkommt und sieht so den geistigen Anspruch an die informatische Bildung an Schulen gefährdet.

Diesen Ausführungen von Baumann [2009a,b] entgegnet Romeike [2010] mit der Praxis an Schulen. Romeike stellt fest, dass für *scratch* zahlreiche und vor allem mannigfaltige Erfahrungsberichte, erfolgreiche Beispiele und Konzepte existieren, wohingegen für *Squeak Etoys* nur wenige Beispiele in der Literatur zu finden sind und diese zumeist nur wiederkehrende Themen aufgreifen (z.B. das Steuern von Au-

[5]Eine Adaption der *Logo Turtle* für den *OLPC XO-Laptop* (http://one.laptop.org/ zuletzt besucht am 15. April 2012.)

tos oder das Nachstellen bestimmter naturwissenschaftlicher Phänomene) und die darüber hinaus auch noch stark von Lehrenden angeleitet werden müssen. Des Weiteren beschreibt Romeike, dass *Squeak Etoys* gerade für Novizen zu viele Fragen zu Beginn offen lässt. Eine zusätzliche Hürde sieht Romeike in dem Ansatz von *Squeak*, dass alle Elemente der Entwicklungsumgebung gleichzeitig veränderbare Objekte sind und so bei unerfahrenen Nutzern unwiderruflich „kaputte“ Zustände dieser hervorbringen kann. Die Eingeschränktheit von *scratch* ist in diesem Fall laut Romeike vorzuziehen, um es Lernenden zu ermöglichen, in einer sicheren und komfortablen Umgebung Programmierkonzepte zu durchdringen und weder auf Syntax, noch auf Systemeigenschaften achten zu müssen. Er sieht darüber hinaus *scratch* im Vorteil, da es von einer Forschungsgruppe entwickelt wird, die auch auf die Bedürfnisse der Nutzer eingeht, dabei jedoch auch immer abwägt und zur Diskussion stellt, ob das Hinzufügen von bestimmten geforderten Eigenschaften eine intuitive und anfängerfreundliche Umgebung gefährden kann. Der Behauptung, *scratch* sei älteren Schülern nicht würdig, entgegnet Romeike zum einen mit Studien, die darauf verweisen, dass *scratch* positive Auswirkungen auf das Erlernen weiterführender Programmiersprachen hat. Zum anderen führt er an, dass *scratch* konstruktionistische Herangehensweisen fördert, da Schüler eigene Ideen umsetzen können, und nicht wie bei *Squeak Etoys* oder auch *TurtleArt* auf die Vorgaben und Hilfe der Lehrenden angewiesen sind, zumal es sich bei den meisten Beispielen für diese Umgebungen laut Romeike zumeist um Aufgaben handelt, die von Schülern als „sinnlos“ und/oder „zusammenhanglos“ angesehen werden. Der Auffassung, dass es *scratch* an Abstraktionsmöglichkeiten fehlt, begegnet Romeike mit dem Hinweis, dass eine spätere Verwendung anderer Programmierumgebungen im Unterricht durchaus beabsichtigt ist. Darüber hinaus verweist er auf *Build Your Own Blocks* (BYOB)[6], eine Adaption von *scratch*, die an der *University of California Berkeley* in Einführungsveranstaltungen eingesetzt wird. BYOB erlaubt diese geforderten Abstraktionen und ermöglicht es, die erstellten Skripte auf Quellcode-Ebene zu betrachten bzw. zu editieren.

Diese Diskussion breitete sich auch im November 2010 über die deutsche *Squeak*-e.V. Mailingliste[7] aus und wurde dort über mehrere Tage hitzig geführt. Sie lässt sich dahingehend zusammenfassen, dass sich die meisten Beteiligten darin einig sind, dass *Squeak* ein umfassendere Umgebung ist, die ihre Vorteile besonders in ihrer Offenheit hat, so dass Nutzer immer auch zu Programmierern werden können. Gleichzeitig ist

[6] http://byob.berkeley.edu/, zuletzt besucht am 15. April 2012.

[7] http://forum.world.st/Etoys-Squeak-vs-Scratch-Diskussionsbeitrag-von-Ralf-Romeike-td3027619.html, zuletzt besucht am 15. April 2012.

die Geschlossenheit und Strukturiertheit von *scratch* vorteilhaft für jüngere Schüler, so dass es sinnvoll erscheint mit *scratch* zu beginnen und dann zu einem späteren Zeitpunkt (10. Klasse) *Squeak* zu verwenden. Uneinigkeit in den Beiträgen war jedoch in der Alltagsrelevanz von *Squeak* zu erkennen. Während *scratch* Programmierkonzepte vermittelt, ist *Squeak* nämlich eine vollwertige Programmiersprache, deren Syntax zu erlernen ist und nur wenig mit aktuellen Programmiersprachen wie C++, Java oder Python gemein hat.

Schlussendlich lässt sich zu diesem Streitgespräch festhalten, dass sich *scratch* aufgrund des breiten und erfolgreichen Einsatzes an Schulen durchgesetzt zu haben scheint. Dies ist auf die Einfachheit der Oberfläche zurückzuführen, die es Lernenden ermöglicht, eigene Ideen umzusetzen und auf diesem Wege informatischen Konzepten zu begegnen. Zu beachten ist dabei aber, dass es nicht das erklärte Ziel von *scratch* ist, eine vollwertige Programmierumgebung anzubieten. Es ist durchaus angedacht, in weiteren Schritten das Konzeptwissen auf eine vollwertige *professionelle* Programmiersprache zu transferieren. Für einen leichten Übergang seien *BYOB* und *Greenfoot* genannt.

### 5.2.2 Greenfoot

Die Verwendung von *Greenfoot* in einem Projekt diente, wie bereits erwähnt, der Überprüfung von Beobachtungen, die in Projekten mit *scratch* gemacht wurden, daher wird *Greenfoot* nur knapp in den Grundfunktionen vorgestellt. Der Hauptunterschied zu *scratch* besteht darin, dass Java als Programmiersprache verwendet wird und damit Quelltext durch die Nutzer bearbeitet werden muss. Für detailliertere Informationen empfiehlt sich der Artikel von Kölling [2010].

*Greenfoot* ist zur Zeit neben *scratch* die einzige Entwicklungsumgebung[8], die weitläufig eingesetzt wird, um Jugendlichen im Anfangsunterricht das Programmieren näher zu bringen [Al-Bow et al., 2008; Fincher und Utting, 2010]. *Greenfoot* ist konzipiert, um programmiertechnische Fragen möglichst zeitig zu vermitteln. Es ist eine Java-Entwicklungsumgebung für alle gängigen Betriebssysteme (Windows, Mac OS X, Linux). Die *Greenfoot*-Oberfläche existiert ebenso wie *scratch* in meh-

[8]Streng genommen wird in diesem Zusammenhang auch immer *Alice* genannt. Dabei handelt es sich jedoch um eine Entwicklungsumgebung für dreidimensionale Szenarien, was deutlich die Komplexität und damit die Altersstufe erhöht. Auch die Entwickler von *Alice* sind der Meinung, dass Vorkenntnisse von Vorteil sind, und sehen *Alice*, auch wenn es Ausnahmen mit jüngeren Projektgruppen gibt, als gutes Werkzeug für einführende Veranstaltungen des Informatik-Studiums [Utting et al., 2010, S. 2 f.]. Aktuelle Betrachtungen raten daher von einem breiten Einsatz im Anfangsunterricht an Gymnasien ab (vergleiche dazu Engbring [2011]).

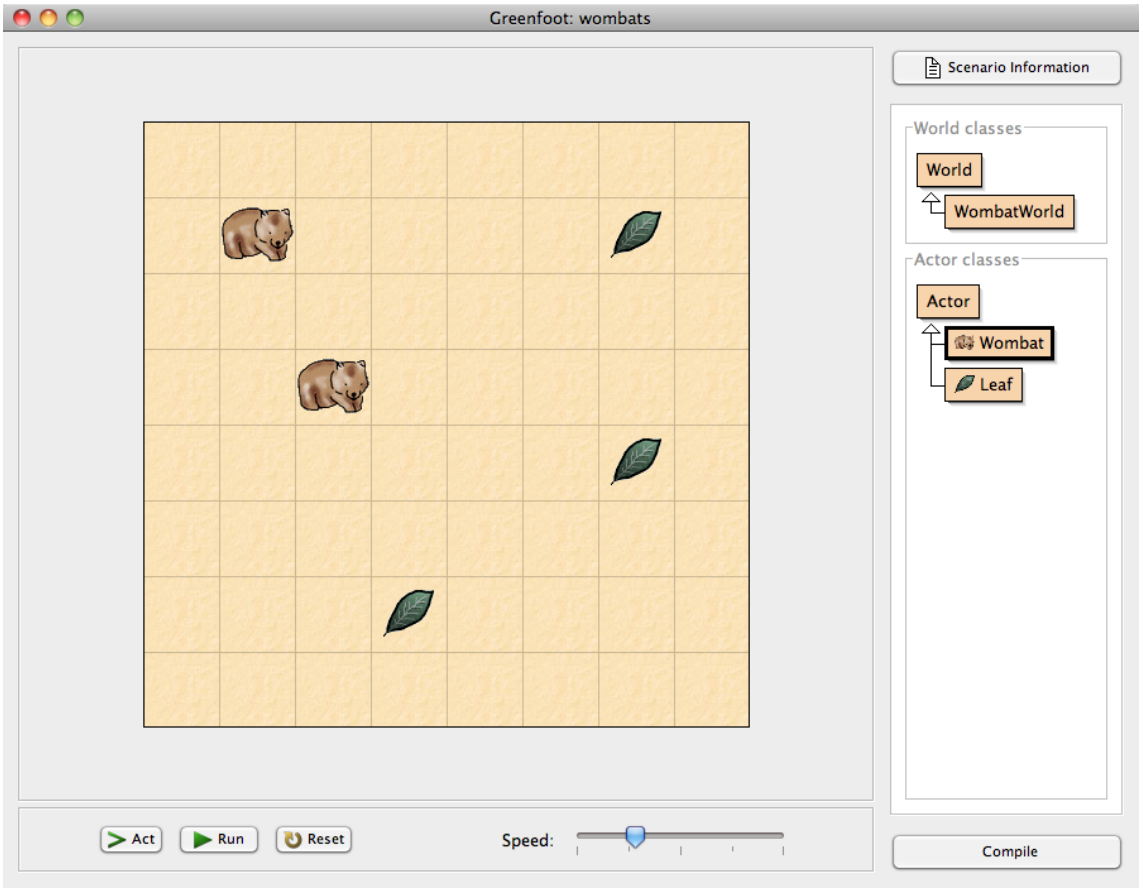



Abbildung 5.5: Das Szenariofenster von *Greenfoot* 2.0.1 für MacOS. Zu sehen ist das *wombats*-Szenario, das zur Grundinstallation von *Greenfoot* gehört.

reren Sprachen, darunter auch Deutsch. Zu beachten ist, dass Java an die englische Sprache angelehnt ist und auch die API[9] zunächst einmal nur in Englisch präsentiert. Es bedarf daher dem Aufwand des Lehrers oder Dozenten, das *Greenfoot*-Szenario dahingehend anzupassen, dass eine deutsche API verwendet werden kann und bei Bedarf, soweit möglich, die zu erstellenden Befehle im Quelltext in Deutsch verfasst werden können. Dies wäre aber befremdlich, wenn man bedenkt, dass Programmiersprachen nie lokalisierte Versionen anbieten und sich somit der häufig genannte Vorteil des Bezugs zu Java zumindest in Teilen aufhebt.

Die Grundidee von *Greenfoot* ist, zweidimensionale Szenarios auf einer Bühne zu präsentieren und anhand dieser, getreu dem *objects first* Ansatz und der *direct state manipulation*, aus Klassen Exemplare zu erzeugen, die mittels Methoden auf der Bühne agieren [Kölling, 2010]. Jugendliche können mit *Greenfoot* auf mehreren

[9] API steht für *application programming interface.*, also eine Programmierschnittstelle.

Ebenen die objektorientierte Programmierung (OOP) durchdringen und es wird daher auch weiterhin an Schulen und Hochschulen zu finden sein.

*Greenfoot* besitzt im Wesentlichen zwei Fenster: Das Szenario (siehe Abbildung 5.5) und den Editor (siehe Abbildung 5.6). Standardmäßig wird beim Öffnen eines Projekts das Szenariofenster gezeigt, das Editorfenster kann zu jeder Klasse manuell geöffnet werden (durch einen Doppelklick auf die jeweilige Klasse oder über das Kontextmenü mittels der rechten Maustaste).

Im Szenario bekommt man auf der rechten Seite in Klassenbäumen einen Überblick, welche Klassen zur Verfügung stehen. Dort kann man Exemplare erzeugen, die dann interaktiv auf der Bühne, welche den größten Raum des Fensters einnimmt, abgesetzt werden können. Mittels verschiedener Knöpfe ist es dem Nutzer möglich, an allen vorhandenen Exemplaren die Methode „Act“ aufzurufen oder das Szenario auf der Bühne ablaufen zu lassen (wiederkehrende „Act“-Aufrufe).

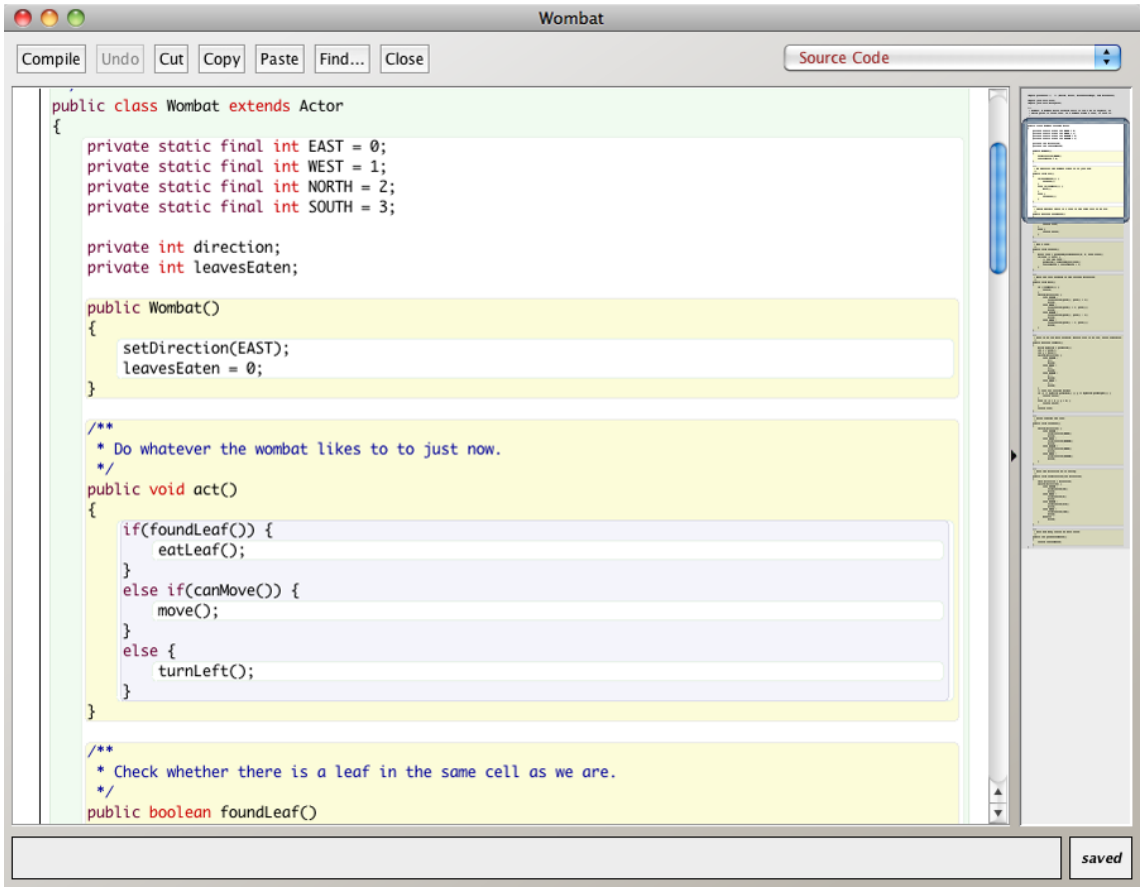



Abbildung 5.6: Das Editorfenster von *Greenfoot* 2.0.1 für MacOS, das die *wombats*-Klasse darstellt. In diesem Fall ist auch zu erkennen, dass der Quelltext zumindest grundlegende Englischkenntnisse erfordert.

Der Editor ermöglicht es, den Quelltext einzusehen, zu ändern oder neu zu schreiben. Die Blöcke und Elemente des Quelltexts werden kontextsensitiv farblich hervorgehoben. Es gibt zwei Ansichten: Die Source Code Ansicht zeigt den eigentlichen Quelltext, die Interface Ansicht stellt die Schnittstelle der Klasse dar. Eine Schaltfläche ermöglicht das Übersetzen der Klasse, *copy & paste* und eine Suche. Die Szenarios sind *microworlds* (vergleiche auch [Papert, 1980]), die von Dozenten vorbereitet und angeboten werden.

Ebenso wie *scratch* verfügt *Greenfoot* über die Möglichkeit, Ergebnisse auf eine Webseite hochzuladen und diese dann im Webbrowser abzuspielen [Henriksen et al., 2010] (vergleiche auch Seite 65 f.). Das Hochladen aus *Greenfoot* heraus erfolgt über die Export-Funktion im „Scenario"-Menü. Im Wesentlichen stimmen die Funktionen der Webplattform mit der von *scratch* überein. Es bestehen jedoch zwei grundsätzliche Unterschiede: Die Galerie ist eine eigene Webpräsenz[10], also losgelöst von der *Greenfoot*-Seite. Auf der Startseite werden neben vorgestellten Projekten Diskussionsaktivitäten besonders hervorgehoben (siehe Abbildung 4.3, S. 67). Eine Sprachauswahl ist nicht möglich (Stand 15. April 2012).

Bezogen auf die Matrix aus Kapitel 4.2.2 ist auch *Greenfoot* darauf ausgelegt Individuen in der Gestaltung zu unterstützen; diese kann sowohl am selben als auch an entfernten Orten stattfinden.

### 5.2.3 Kontext Schulunterricht

Der Erfolg von *scratch* und *Greenfoot* hängt sicherlich auch von der gewählten Unterrichtsform und dem darin gewählten didaktisch-methodischen Vorgehen ab. Nach Abschnitt 3.2.1 und der Lehrerbefragung in 3.2.2 ist davon auszugehen, dass beide ILEs hauptsächlich im Gruppenunterricht verwendet werden, der jedoch über wenig unterstützende Infrastruktur oder Organisation verfügt. Die Schüler werden demnach in Kleingruppen von maximal drei Personen zusammen arbeiten, eine Kommunikation über die Gruppen hinweg dürfte nicht stattfinden bzw. wäre nicht erwünscht seitens der Lehrer. Es ist zu erwarten, dass weitläufig eine Mischform aus gelenktem Entdecken und freiem Forschen eingesetzt wird. Dieser Kontext wird im Folgenden als klassischer Gruppenunterricht bezeichnet und im gemeinsamen Einsatz mit *scratch* in Kapitel 9.3 wieder aufgegriffen.

[10] http://www.greenfootgallery.org/, zuletzt besucht am 15. April 2012.

## 5.3 Kapitelzusammenfassung

Dieses Kapitel gab einen Überblick über ILEs, die speziell konzipiert sind, um Novizen bzw. Kinder und Jugendliche den Einstieg ins Programmieren einfach zu ermöglichen. In geschichtlichen Betrachtungen wurden Logo und Karel als Vertreter von Systemen vorgestellt, die Programmiersprachen vereinfachen bzw. Programmierkonzepte hervorheben sollen. Ein besonderes Augenmerk bei den historischen Umgebungen wurde auf die Systeme gelegt, die soziale Interaktion einbeziehen.

Im zweiten Teil des Kapitels wurden die momentan erfolgreichen ILEs *scratch* und *Greenfoot* vorgestellt. *Scratch* stellt eine grafische Programmierumgebung dar, die auf einer Bühne (interaktive) Szenarien simuliert. *Greenfoot* ist eine Java-Umgebung, die auch Wert darauf legt, Szenarien zu visualisieren. Darüber hinaus bleibt festzuhalten, dass *scratch* viele Vorteile für Novizen durch die grafische Programmierumgebung bereit hält, gleichzeitig jedoch eine aktive Diskussion geführt wird, ob nicht doch textbasierte trotz ihrer Hürden im Schulkontext vorzuziehen sind. Deutlich mehr Autoren sprechen sich momentan aber für *scratch* als erste Umgebung im Schulkontext aus.

Im nachfolgenden Kapitel werden Projekte mit Schülern vorgestellt, bei denen *scratch* und *Greenfoot* zum Einsatz kamen und beobachtet wurde, wie Gruppen mit einem gemeinsamen Ziel damit arbeiten.

# Kapitel 6

# Projekte im Schulkontext

Dieses Kapitel stellt detailliert alle im Zuge dieser Promotion durchgeführten Projekte mit Schülern vor. Zuvor werden die Herangehensweisen der Projekte didaktisch eingeordnet. Die genannten Beschreibungen der Projekte stützen sich auf handschriftliche Notizen (*research diary*), die während aller Projekte geführt wurden. Diese Aufzeichnungen beinhalten zeitliche Abläufe, Inhalte und Prozessschritte der jeweiligen Projekte. Darüber hinaus dokumentieren sie auch das Verhalten der Teilnehmer, gewählte Lösungswege und Gruppendynamiken.

Grundlegend für alle Projekte mit mehreren Projekttagen war das Ziel, ein rudimentäres Verständnis für die moderne Softwareentwicklung zu vermitteln, indem agile Methoden (*pair programming*, *user stories*, *informative workspace*, *meetings* und *collective code ownership*) aktiv zum Einsatz kommen sollten. Wobei im größten Teil darauf Wert gelegt wurde, freie Projektphasen für die eigentliche Entwicklung anzubieten. Auf diese Weise kann ein Bezug zur realen Berufswelt hergestellt werden (vergleiche Kapitel 3.1 Fisker et al. [2008] und Kapitel 3.1.1 Keefe et al. [2006]). Nach Loftus und Ratcliffe [2005] (Kapitel 3.1.1) wurde während der Konzeption davon ausgegangen, dass eine einzelne Praktik der agilen Methoden nicht ausreicht, um im Schulkontext soziale Aspekte der Softwareentwicklung hinreichend hervorzuheben. Gleichzeitig war die komplette Umsetzung der agilen Methoden als ganzheitliches Vorgehensmodell zu umfangreich für Projekte mit einer maximalen Dauer von fünf Tagen, so dass die fünf genannten Praktiken als guter Kompromiss erschienen, um Anreize zur Interaktion zwischen den Teilnehmern zu schaffen (vergleiche Schneider und Johnston [2005], Kapitel 3.1.1). Ein Einsatz mehrerer Praktiken der agilen Methoden ist ebenfalls sinnvoll, um frühzeitig zu verhindern, dass Schüler eigene Arbeitsprozesse herausbilden, die den agilen Methoden entgegenstehen, so dass sie zu

einem späteren Zeitpunkt nicht mehr bereit wären, den Mehraufwand dafür in Kauf zu nehmen (vergleiche Rico und Sayani [2009] und Schild et al. [2010], Kapitel 3.1.1). Die getroffenen Maßnahmen waren konzipiert, um einen größeren Bekanntheitsgrad der agilen Methoden zu fördern (vergleiche Kapitel 3.1.1)

## 6.1 Hintergrund Didaktik der Informatik

Zunächst ist es wichtig, die nachfolgend vorgestellten Projekte in Beziehung zur Didaktik der Informatik zu setzen. Die Wahl fiel auf projektartige Herangehensweisen, da diese Form geeignet ist, um die in Kapitel 4.1.1 genannten Idealvorstellungen von Papert [1980] zum gemeinsamen Lernen ohne Hierarchien und gegenseitige Hilfestellungen umzusetzen. Auch der Gruppengedanke und besonders der Einsatz von *pair programming* fußt auf den Gedanken von Papert, dass Lernen immer in Interaktion mit anderen Lernenden geschehen und von Kooperation geprägt sein sollte. Auch Fehler oder Probleme können laut Papert besser in der Gruppe aufgefangen werden. Gleichzeitig kann durch eine Gruppenstruktur und ein gemeinsames Projekt versucht werden, den Bedenken von Turkle [1984] entgegenzuwirken, dass Computerlernumgebungen schnell zu einer Isolierung der Lernenden führen.

Bei der Konzipierung und Durchführung der Projekte war der Gedanke grundlegend, dass es sich um Anfangsunterricht handeln sollte, da dort davon auszugehen ist, dass die Schüler noch „ungeprägt“ und „unvoreingenommen“ sind [Schubert und Schwill, 2011, S. 287] und somit noch von der Attraktivität der Informatik durch Bezüge zur professionellen Softwareentwicklung überzeugt werden können. Auch Hubwieser [2007] macht darauf aufmerksam, dass frühestens in der sechsten Klasse (ab ca. 11–12 Jahren) mit einem Fundamentum der Informatik begonnen werden sollte, jedoch auch nicht sehr viel später, da sonst zu befürchten wäre, dass Schüler durch eine „häusliche Auseinandersetzung“ zu einem einseitigen Bild der Informatik bzw. der darunter liegenden Konzepte gelängen und diese dann nur wieder schwer zu korrigieren sind [Hubwieser, 2007, S. 100 f.]. Auch sieht er in dieser Altersstufe die beste Chance, gleichermaßen auf Teilnehmer mit und ohne Vorkenntnisse einzugehen und sowohl Mädchen als auch Jungen anzusprechen. Dies stützt die Beobachtungen aus Kapitel 3.1.1 und der daraus resultierenden Behauptung aus Kapitel 3.2, dass der Einsatz von *pair programming* oder ganzheitlichen agilen Methoden in der Universitätsbildung zu spät greift, um auf soziale Interaktion in der Softwareentwicklung hinweisen zu können.

Ein Anfangsunterricht oder auch Fundamentum besitzt immer Komponenten, die zu bedenken sind, weil sie erschwerend wirken können [Schubert und Schwill, 2011, S. 287]:

- Es werden heterogene Schülergruppen (Computernovizen bis hin zu Teilnehmern mit großen Vorkenntnissen) angesprochen.
- Die Vorerfahrungen der Schüler führen meist zu Selbstüberschätzung und/oder mangelnder Kooperationsbereitschaft.
- Verschiedene Intentionen der Teilnehmer abhängig vom Kenntnisstand (Novizen: Allgemeiner Einblick Informatik. Teilnehmer mit Vorkenntnissen: Vermittlung von tiefergehenden Tricks zur Verwendung eines Computersystems) sind zu erwarten.
- Unzureichende Ausstattung an Schulen ist zu beachten, da diese sich gerade bei Teilnehmern mit Vorkenntnissen negativ auf die Motivation auswirken kann.
- Unterschiedliche Rahmenbedingungen sind je nach Bundesland gegeben.

Die beiden letzten Punkte wurden nicht weiter berücksichtigt, da die angebotenen Projekte von der Universität aus organisiert wurden. Damit genossen sie den Vorteil, zum einen nicht an Vorgaben der Bundesländer gebunden zu sein, und zum anderen konnten, wenn nötig, universitätseigene, aktuelle Geräte in den Projekten verwendet werden.

Schubert und Schwill beschreiben einen konkreteren Aufbau für den Anfangsunterricht und beziehen sich dabei auf [Koerber und Peters, 1995]:

- Didaktische Reduktion auf typische Anwendungsszenarien.
- Offene Probleme, die fordernd und ausreichend komplex sind, jedoch in jedem Fall in mehreren Schritten lösbar sind.
- Raum für eigene Herangehensweisen.
- Leichter Einstieg mit minimalem Bezug zu Vorkenntnissen.
- Abdeckung von gesellschaftlichen Bereichen mit Möglichkeiten zur erfahrbaren Realität der Teilnehmer.
- Angebot von Sachinformationen über das eigentliche Lernziel hinaus.

Hubwieser [2007] dagegen spricht davon, dass ein Anfangsunterricht möglichst „spielerisch und handlungsorientiert, keinesfalls aber ungenau oder unsystematisch“ sein sollte [Hubwieser, 2007, S. 101]. Er formuliert jedoch allgemeine Forderungen an den Unterricht und deckt darin ähnliche Aspekte wie Schubert und Schwill [2011] ab. Er fordert zusätzlich zu den genannten Punkten eine entspannte Arbeitsatmosphäre mit einer altersgemäßen Darstellung der Inhalte, die eine aktive Auseinandersetzung mit dem Stoff erlaubt [Hubwieser, 2007, S. 67].

Projektartige Angebote eignen sich, um die oben genannten Forderungen an den Unterricht nach Hubwieser [2007] abzudecken. So verweist Hubwieser auch darauf, dass „wissenschaftliche und industrielle“ Aufgaben in der Praxis vermehrt in Gruppenarbeit gelöst werden und daher Schüler in gesondertem Maß durch Gruppenarbeit darauf vorbereitet werden sollten [Hubwieser, 2007, S. 37]. Gerade Gruppenarbeit erscheint besonders passend für projektartige Angebote. Bei der Begründung für Gruppenarbeit von Hubwieser ist auffällig, dass keine Verweise auf aktuelle Strömungen der Softwareentwicklung existieren, obwohl dort bereits Gruppenarbeit stark strukturiert und damit auch gefördert wird. Das Fehlen dieser nützlichen Verweise auf reale Berufsbilder wurde bereits in Kapitel 3 als Faktor identifiziert, der zu einem einseitigen Bild der Informatik beitragen kann. Schubert und Schwill [2011] beschreiben Projekte im Schulkontext noch etwas expliziter und sprechen sich noch eindeutiger für die projektbasierte Unterrichtsform in der Informatikbildung aus [Schubert und Schwill, 2011, S. 303]. Sie warnen allerdings unter Berufung auf Baumann [1996] davor, den Projektbegriff der Informatik bzw. Softwareentwicklung mit dem der Schulinformatik gleichzusetzen [Schubert und Schwill, 2011, S. 308 f.], da in der Softwareentwicklung der Fokus auf einer effiziente Verteilung der Aufgaben liege, wohingegen Projekte im Unterricht gerade das gemeinsame Erarbeiten und Kommunizieren fördern solle. Hier sei dahingestellt, ob Praktiken aus den agilen Methoden wirklich dem geforderten Grundgedanken eines schulischen Projekts widersprechen (vergleiche Kapitel 2.2).

Konkret ergab sich für die Planung der durchgeführten Projekte, dass sie sich primär an eine Altersgruppe zwischen 11 und 15 Jahren richten sollte, die in kurzen Projekten möglichst spielerisch und selbstgesteuert digitale Geschichten oder Computerspiele entwerfen sollte. In den Projekten wurden inhaltliche Fragestellungen behandelt, die für Jugendliche relevant sind. Als ein solches Themenfeld wurden interkulturelle Fragestellungen identifiziert, da diese aufgrund des stark steigenden Anteils von Schülern mit Migrationshintergrund laut Rahmenplan fächerübergreifend in allen Hamburger Schulen verankert sind. Auch andere Forschungsberichte ha-

ben wichtige Beweggründe, wie beispielsweise die Gefahr des *digital divides*[1], für die Fokussierung auf dieses Themenfeld erkannt [Schubert et al., 2011]. In einem Interview mit dem Religionslehrer Andreas Gloy (Gymnasium Kirchdorf/Wilhelmsburg), der interkulturelle Bildung an seiner Schule erfolgreich konzeptionell betreut, kam zum Vorschein, dass dieses Thema häufig recht starr in den einzelnen Fächern angewendet wird und es den Schülern meist nicht frei steht, eigene Geschichten zu dem Thema zu erarbeiten. Für detailliertere Einblicke zur Entwicklung von interkulturellen Computerspielen mit Jugendlichen sei auf Göttel [2009a] verwiesen[2].

Man kann davon ausgehen, dass die Gestaltung von Geschichten oder Spielen eine offene Problemstellung repräsentiert, die die Möglichkeit bietet, sich je nach Kenntnisstand und Interesse weiter in die Problematik einzuarbeiten, aber auch zu einem wohldefinierten Ende zu gelangen. Die Verwendung von Methoden aus der modernen Softwareentwicklung sollte eine Infrastruktur bieten, die ein strukturiertes Vorgehen besonders im Gruppenkontext vereinfacht und es erlaubt, auf unterschiedliche Kenntnisstände zu reagieren. Die entsprechenden Empfehlungen wurden zu Beginn der Projekte genannt, jedoch nicht aktiv während der freien Projektphase eingefordert, da nach Harel [1991] (vergleiche Kapitel 4.1.1) davon ausgegangen werden kann, dass Jugendliche – wie von Papert [1980] vermutet – in der Lage sind, eigene Ziele und Herangehensweisen herauszuarbeiten und beizubehalten.

Die Wahl des Programmierwerkzeugs fiel auf *scratch*, da dies einen möglichst einfachen Einstieg ermöglicht und trotzdem genügend Möglichkeiten bietet, um komplexe Abläufe beschreiben zu können (vgl. Kapitel 5.2). Zu einem späteren Zeitpunkt wurde ein Projekt mit *Greenfoot* durchgeführt, das häufig als weiterführender Schritt nach *scratch* empfohlen wird. Die Verwendung von *Greenfoot* diente als Kontrolle, um die Beobachtungen zu Projekten mit *scratch* und die daraus resultierenden Anforderungen mit einem weiteren ILE abgleichen zu können.

## 6.2 Durchgeführte Projekte

Im Laufe dieser Arbeit wurden im Zeitraum 2007–2011 acht Projekte mit insgesamt 75 Schülern im Alter von 10–17 Jahren durchgeführt, wobei der Hauptanteil der

[1] Die Annahme, dass bestimmten Gesellschaftsgruppen (zumeist mit Migrationshintergrund) der Zugang zu Kommunikationstechnologien mangels gezielter Förderung verwehrt bleibt, wird als digitale Kluft bezeichnet.

[2] Hierzu existiert ebenfalls eine vom Autor entworfene frei zugängliche Unterrichtseinheit unter http://www1.fh-koeln.de/spielraum/level3/schulische_medienpaedagogik/00567/index.html, zuletzt besucht am 15. April 2012.

Teilnehmer bei 11–14 Jahren lag. Es handelte sich immer um Projekttage oder Projektwochen, so dass die Schüler keinen weiteren schulischen Verpflichtungen nachkommen mussten. Beim Schnupperstudium und den Girls' Days existierte jedoch ein Rahmenprogramm, welches das Studium der Informatik bewarb. Einen umfassenden Überblick zu den einzelnen Projekten gibt Tabelle 6.1.

| Art | ILE | Dauer (in Tagen) | Teilnehmer / davon weiblich | Alter (Jahre) |
|---|---|---|---|---|
| Schnupperstudium | *scratch* | 5 | 12 / 3 | 12–16 |
| Projektwoche KiWi | *scratch* | 5 | 5 / 0 | 12–17 |
| Projektwoche HDJ | *scratch* | 3 | 6 / 6 | 15–17 |
| Projektwoche NSG | *scratch* | 2 | 12 / 6 | 11–13 |
| Girls' Day | *Greenfoot* | 1 | 14 / 14 | 10–15 |
| 2 x Girls' Day | *scratch* | je 1 | je 12 / 12 | 10–15 |
| Begleittermine Informatik AG | *scratch* | 3 | 12 / 4 | 11–13 |

Tabelle 6.1: Durchgeführte Projekte mit Jugendlichen 2007–2011 (KiWi = Gymnasium Kirchdorf/Wilhelmsburg, HDJ = Haus der Jugend, NSG = Niels-Stensen-Gymnasium), die als Beobachtungsgrundlage für die Entwicklung des Prototyps und für die Anpassung der agilen Methoden dienten (siehe Kapitel 8). Auch die vorgestellten Personas aus Kapitel 9 basieren auf den Beobachtungen zu diesen Projekten.

### 6.2.1 Schnupperstudium

Das erste Projekt wurde im Rahmen des Schnupperstudiums 2007 an der Universität Hamburg auf dem Informatik-Campus durchgeführt. Es handelt sich beim Schnupperstudium um ein einwöchiges Angebot des Fachbereichs Informatik an alle Schüler der Klassenstufen 10–13 aus dem Großraum Hamburg. Neben der Projektarbeit werden eine Vorlesung, Demos, Berufseinblicke und Studienberatungsgespräche angeboten, am Nachmittag des letzten Tags ist eine Präsentation aller Gruppenergebnisse vorgesehen (für einen Überblick über den entsprechenden Stundenplan siehe Abbildung 6.1). Daraus ergab sich für die Projektarbeit eine effektive Zeit von ungefähr 13 Stunden. Von den zwölf Teilnehmern zwischen 12 und 16 Jahren waren drei weiblich. Somit ergaben sich drei Gruppen zu je vier Teilnehmern, wobei jede Gruppe eine weibliche Teilnehmerin aufwies. Als Entwicklungsumgebung wurde *scratch* verwendet, des Weiteren wurde ein CommSy-Raum angeboten, um jeglichen Austausch innerhalb der Gruppen zu organisieren. Im CommSy wurde auch eine abschließende Evaluation angeboten.

**Stundenplan 2007** (Pausen bzw. "akademische Viertel" nicht dargestellt):

| | Montag 22.10. | Dienstag 23.10. | Mittwoch 24.10. | Donnerstag 25.10. | | Freitag 26.10. |
|---|---|---|---|---|---|---|
| 09:00 | Willkommen Ziele | Inhalte und Methoden der Informatik | Übersicht Studiengänge | Projektarbeit | Eignungsberatung in Gruppen | Projektarbeit |
| 09:30 | | | | | | |
| 10:00 | Probevorlesung: Interactive Visual Computing (IVC) | Gespräch mit Berufstätigen | Projektarbeit | | | |
| 10:30 | | | | | | |
| 11:00 | | Projektarbeit | | | | Projekt-Präsentationen |
| 11:30 | | | | | | |
| 12:00 | Projektvorstellung Projektauswahl | | | | | |
| 12:30 | | | | | | |
| 13:00 | Mittagessen | Mittagessen | Mittagessen | Mittagessen | | Mittagessen |
| 13:30 | | | | | | |
| 14:00 | Projektarbeit | Forschungsdemos | Forschungsdemos | Forschungsdemos | | Projekt-Präsentationen |
| 14:30 | | Projektarbeit | Demos Studentischer Projekte | Projektarbeit | | |
| 15:00 | | | | | | Rückblick, Feedback |
| 15:30 | | | | | | |

| Legende der Farben: | Vortrag / Plenum | Gespräche, Diskussionen | Vorführungen | Arbeit in den Projektgruppen | Abschlußpräsentationen der Projektgruppen |
|---|---|---|---|---|---|

Abbildung 6.1: Der Stundenplan des Informatik Schnupperstudiums der Universität Hamburg 2007.

### Ablauf

Am ersten Tag wurden Kennenlernspiele durchgeführt, Diskussionen über Computerspiele angeregt und eine kurze Einführung in Ideen der iterativen Softwareentwicklung und XP gegeben. Abschließend wurden die Teilnehmer in Gruppen eingeteilt und gebeten, in einer einstündigen Brainstorming-Einheit eine grobe Spielidee zu einer interkulturellen Begegnung zu entwickeln. Die Gruppeneinteilung erfolgte nach genannten Interessen am Game Design[3] der Teilnehmer in der Vorstellungsrunde. Es wurde darauf geachtet, dass jede Gruppe aus Teilnehmern mit verschiedenen Interessen besteht, um alle Bereiche des Game Design zumindest theoretisch abzudecken. Am zweiten Tag wurden Konzepte des XP (pair programming, informative workspace, meeting und stories) vorgestellt und erklärt, wie sie in der restlichen Projektzeit verwendet werden sollten. Eine Einführung in *scratch* und CommSy stellte den Auftakt zur eigentlichen Programmierarbeit dar. Die folgenden drei Tage waren als freie Projektarbeit mit der Randbedingung konzipiert, dass am Anfang jedes Tages ein *meeting* durchgeführt wurde, bei dem jeder Teilnehmer der gesamten Gruppe seine bisherigen Ergebnisse und die aktuellen Vorhaben vorstellen sollte. Eine detaillierte Auflistung der genannten Punkte findet sich in Tabelle 6.2 wieder. Am Ende des Projekts wurden im CommSy-Raum Diskussionen durch den Autor angeregt, um eine Feedbackmöglichkeit anzubieten. Dort konnten Gründe genannt werden,

[3]Zur Auswahl standen Story, Grafik oder Technologie.

warum *scratch* „super“ bzw. „doof“ ist und was am interkulturellen Hintergrund „spannend“ bzw. „langweilig“ ist.

| Zeiteinteilung | Tag 1 (120 min) |
|---|---|
| Begrüßung | ca. 5 min |
| Kennenlernen | ca. 35 min |
| Einführung Softwareentwicklung | ca. 20 min |
| Brainstorming | ca. 60 min |
| Zeiteinteilung | Tag 2 (195 min) |
| Einführung XP | ca. 10 min |
| Einführung *scratch*/CommSy | ca. 30 min |
| freie Projektarbeit | ca. 155 min |
| Zeiteinteilung | Tag 3/4/5 (Summe) |
| freie Projektarbeit | ca. 155 min |

Tabelle 6.2: Der Ablaufplan des Projekts beim Schnupperstudium 2007.

### Beobachtungen

Während der Gruppeneinteilung und der Brainstorming-Phase war unter den Teilnehmern häufig eine Unsicherheit in der Kommunikation in Gruppen zu verspüren; dies führte zu gehemmt geführten Brainstormingprozessen und recht einfachen und wenig inspirierten Spieleideen. Diese Ideen wurden erst während der Programmierphase ernsthaft ausgearbeitet und verfeinert. Diese Unsicherheit in der Gruppe zeigte sich auch in der Kennenlernrunde, bei der die Teilnehmer unter anderem gebeten wurden, das schlechteste Computerspiel zu nennen, das sie je gespielt hatten. In dieser Altersklasse zeigten sich häufig Probleme im Mitteilen von ungewöhnlichen Ansichten oder fehlerhaften Versuchen bzw. dem Umgang damit.

Schon am ersten Tag gab es einzelne Teilnehmer, die bestimmte nicht abgeschlossene Arbeitspakete alleine zuhause fertigstellen wollten. Es handelte sich dabei um zwei männliche Personen, die über die Woche verteilt immer wieder bestimmte Teilaufgaben alleine und zum nächsten Tag lösen wollten. Diese und noch zwei weitere Teilnehmer (alle männlich) beschwerten sich gerade zu Beginn über *scratch*, da es sich dabei ihrer Ansicht nach nicht um eine echte Programmierumgebung handelte. Im Verlaufe des Projekts stellten jedoch zwei fest, dass mit *scratch* doch auch komplexere Programmstrukturen erschaffen werden können bzw. es sich dabei um ein adäquates Werkzeug für den Zeitraum einer Woche handelt. Die anderen beiden blieben bei ihrer Meinung, produzierten jedoch ein Ergebnis, das keine kom-

plexere Programmiersprache rechtfertigen würde. Viele angeblich nicht vorhandene Möglichkeiten sind in *scratch* vorhanden (z.B. Dateiimport), wurden aufgrund der ablehnenden Haltung nicht von diesen Teilnehmern entdeckt. Ein weiteres Indiz für die Unangemessenheit ihrer Ablehnung ist, dass sie in ihrem Programm viel Musik eingebaut haben und die einfache Einbindung für selbstverständlich erachteten. Betrachtet man dagegen die von ihnen geforderten vollwertigen echten Programmiersprachen, wie z.B. C++, so erkennt man, dass diese Teilnehmer den großen Aufwand, der mit so einer Programmiersprachen einhergeht, falsch einschätzten: Die korrekte Integration bzw. Ausgabe von Musik- und Toneffekten in einer Sprache wie C++ hätte die Verwendung von Musik und Tönen in der geringen Projektzeit sicherlich erschwert, wenn nicht gar verhindert.

Am zweiten Tag ereignete sich der Fall, dass in zwei Gruppen je ein Teilnehmer krankheitsbedingt fehlte und sich so in Folge in einer Gruppe ein Paar bildete, das absolut nicht gewillt war, andere an ihren Plänen teilnehmen zu lassen. Die dritte Teilnehmerin hatte keinerlei Möglichkeiten, mit diesen beiden in Kontakt zu treten, eine Kommunikation oder Aufteilung der Aufgaben war nicht möglich. Nach mehreren Vermittlungsversuchen wurde entschieden, diese Teilnehmerin der anderen gut kommunizierenden Gruppe mit aktuell nur drei Teilnehmern zuzuteilen. Bis zum Ende des Projekts war es trotz mehrmaliger und verschiedener Versuche keinem Teilnehmer oder Betreuer möglich, einen ordentlichen Kontakt zu dem übrig gebliebenen Zweierteam aufzubauen (auch der wieder gesundete Teilnehmer wurde an Tag 3 einer anderen Gruppe zugeteilt). Es bleibt jedoch zu sagen, dass dieses Zweierteam ein voll funktionierendes und positiv aufgenommenes Adventure Game zum Abschlusstermin präsentierte und somit zumindest die Anforderung erfüllen konnte, die ein lauffähiges Programm zum Projektabschluss vorsah. In einer anderen Gruppe gab es einen ähnlichen Fall, bei dem es nicht gelang, die weibliche Teilnehmerin in die Gruppe zu integrieren. Sie verlor sich vollkommen in der Erstellung von immer neuen linearen multimedialen Kurzgeschichten, die sie jedoch nie den anderen Gruppenteilnehmern vorstellte. Auch hier gelang es nicht, Kooperation herzustellen, so dass sich die Gruppe intern darauf verständigte, dass die Teilnehmerin für die *Quality Assurance* zuständig war (diese Gruppe hatte sich intern bereits verschiedene Aufgaben entsprechend der professionellen Spieleentwicklung zugeteilt), was bedeutete, dass sie die verschiedenen Spieleprototypen der anderen drei Teilnehmer zu Testzwecken durchspielen sollte.

Zwischenmenschliche Kommunikation (wie z.B. Vorträge, kurze Erläuterungen oder auch Gruppendiskussionen) wurde immer wieder stark von den vorhandenen

und eingeschalteten Monitoren bzw. PCs beeinflusst, da diese häufig anziehend auf die Teilnehmer wirkten, so dass mit geringer Aufmerksamkeit für den eigentlichen zwischenmenschlichen Austausch zu rechnen war. Der *informative workspace* wurde nur auf Aufforderung, dann jedoch auch rege verwendet, so dass die entstandenen Poster doch als Argumentationsgrundlage für eine weitere Verwendung dieses Konzepts dienen können (siehe Abbildung 6.2).

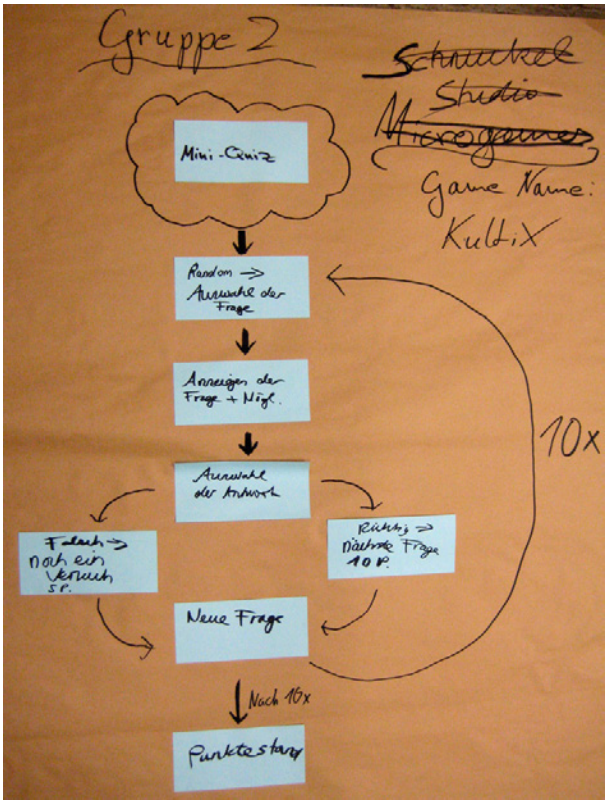



Abbildung 6.2: Ein Poster des Schnupperstudiums.

Der CommSy-Raum wurde hauptsächlich genutzt, um *scratch*-Projekte und Bilder hochzuladen. Jede Gruppe verfügte über eine zentrale Materialkategorie, in die sie ihre Ergebnisse einfügte. Eine Gruppe hatte darüber hinaus weitere Materialien zu bestimmten Kontexten angelegt und diese befüllt. So gab es ein Material für ihre Bilder und einige Materialien, die Vorabversionen oder auch kleine Nebenprojekte enthielten. Die Materialien wurden bis auf eine Ausnahme nicht textlich gekennzeichnet oder erklärt. Die Kommentarfunktion wurde bei den Materialien nicht verwendet. Auf die Initialfrage unter Diskussionen, ob *scratch* „doof“ oder „super“ sei und ob der interkulturelle Hintergrund „spannend“ oder „langweilig“ sei, reagierten nur zwei Teilnehmer. Beide äußerten sich abwägend und glichen sich gegenseitig aus, so dass sie zum einen *scratch* als zu simpel, zum anderen aber als guten Einstieg

benannten. Ein ähnliches Bild ergab sich beim interkulturellen Hintergrund: Ein Beitrag lobte die Möglichkeit zu diesem Thema etwas lernen zu können, der andere fand diese Einschränkung hinderlich. Ankündigungen wurden in diesem CommSy-Raum nicht hinterlegt.

Es gab mehrere Teilnehmer, die nach dem Projekt versicherten, sich nun für ein Informatik-Studium entscheiden zu wollen. Nachweislich schrieb sich ein Teilnehmer im Folgejahr an der Universität Hamburg für ein Studium als Informatik BSc. ein.

### 6.2.2 Projektwoche Gymnasium Kirchdorf/Wilhelmsburg

Dieses Projekt wurde im Rahmen der Projektwoche 2007 am Gymnasium Kirchdorf/Wilhelmsburg (KiWi) in Hamburg angeboten. Es handelte sich um eine schulweite Projektwoche mit freier Projektwahl unabhängig von Alter oder Klassenstufe. Es standen zahlreiche Projekte zur Auswahl. Das hier beschriebene stellte das einzige mit informatischem Inhalt dar und war darüber hinaus das einzige externe Angebot. Zum Projekt meldeten sich fünf männliche Schüler im Alter zwischen 12 und 17 Jahren an. Für das Projekt standen alle fünf Tage zur Verfügung, die Schultage (9h30–12h30) waren ausschließlich für die Projektarbeit vorgesehen, nur der letzte Tag war als Präsentationstag vorgesehen. Somit ergab sich eine effektive Projektzeit von ca. 12 Zeitstunden. *Scratch* diente wieder als Entwicklungsumgebung und ein CommSy-Raum sollte ebenfalls die Gruppe unterstützen. Jedoch war die Schule nicht in der Lage, einen Internetzugang im Projektraum anzubieten, so dass der CommSy-Raum nur zur Benutzung nach der Schule zur Verfügung stand.

#### Ablauf

Aufgrund der großzügigen, täglichen Projektzeiten war es möglich, sämtliche einführenden Aspekte am ersten Tag zu absolvieren und ab dem zweiten Tag eine freie Projekt- und Programmierarbeit in Paaren zu initialisieren (siehe Tabelle 6.3). Auch hier wurde dann in der freien Projektphase nur auf das morgendliche *meeting* Wert gelegt. Diesmal wurden bereits mit dem Beginn des Projekts im CommSy-Raum Diskussionen angelegt, ob *scratch* „super“ bzw. „doof“ und was an dem interkulturellen Hintergrund „spannend“ bzw. „langweilig“ sei. Während des vierten Tags wurden die Teilnehmer noch einmal explizit um Feedback zum Projekt gebeten. Sie konnten während des gesamten Tages ihre Meinungen zu *scratch*, der interkulturellen Thematik, dem Arbeitsaufwand, der Gruppendynamik bzw. -struktur, den eigenen Lernerfolg und einem möglichen Ausblick anonym auf Karteikarten aufschreiben.

Dabei kamen blaue Karteikarten für positive Aspekte und rote Karteikarten für negative Aspekte zum Einsatz.

| Zeiteinteilung | Tag 1 (180 min) |
|---|---|
| Begrüßung | ca. 5 min |
| Kennenlernen | ca. 35 min |
| Einführung Softwareentwicklung/XP | ca. 30 min |
| Einführung *scratch*/CommSy | ca. 30 min |
| Brainstorming | ca. 80 min |
| Zeiteinteilung | Tag 2/3/4 (540 min) |
| freie Projektarbeit | je 180 min |

Tabelle 6.3: Der Ablaufplan des Projekts bei der Projektwoche am KiWi 2007.

### Beobachtungen

Die fünf Schüler waren angetan von der Idee, ein gemeinsames Spiel mit interkulturellem Kontext zu konzipieren. Sie versuchten ganz selbstverständlich, eigene biografische Hintergründe in das Gesamtkonzept einzuarbeiten[4]. So kristallisierte sich während des Brainstormings heraus, dass Levels geschaffen werden sollten, die Allgemeinwissen über deutsche, chinesische und afghanische Kulturen abfragen. Sie kamen zur Spielidee, in der sich der Spieler durch verschiedene Räume bewegen muss, in denen jeweils eine Person aus dem jeweiligen Land Fragen über dieses Land und dessen Kultur stellt. Wurden alle Fragen in einem Raum richtig beantwortet, so erhält der Spieler einen Schlüssel für den nächsten Raum. Diese konkrete Idee nahm jedoch erst am Folgetag nach dem Brainstorming Konturen an und wurde wiederum erst einen Tag später auf dem Poster des *informative workspace* verschriftlicht (siehe Abbildung 6.3). Da es in *scratch* nur ein geringer Aufwand ist, ausgeschnittene Bilder in das Programm einzuarbeiten, waren die Schüler sofort von der Idee begeistert, sich selbst als Charaktere in das Spiel einzubauen. Mit einer Digitalkamera fotografierten sie sich gegenseitig in mehreren Posen, um diese an die entsprechenden Spielsituationen anzupassen (siehe Abbildung 6.4).

Obwohl es fünf Teilnehmer waren, funktionierte das Programmieren in Paaren gut. Sie kamen zu der Lösung, dass die beiden Ältesten (15 und 17 Jahre alt) eine Zweiergruppe bildeten und einer von ihnen noch mit einem Jüngeren ein zweites

[4]Es muss erwähnt werden, dass alle Teilnehmer den zuvor bereits als Interviewpartner erwähnten engagierten und beliebten Religionslehrer hatten, der interkulturelle Bildung bereits vor der Projektwoche ansprechend in seinem Unterricht vermittelte.

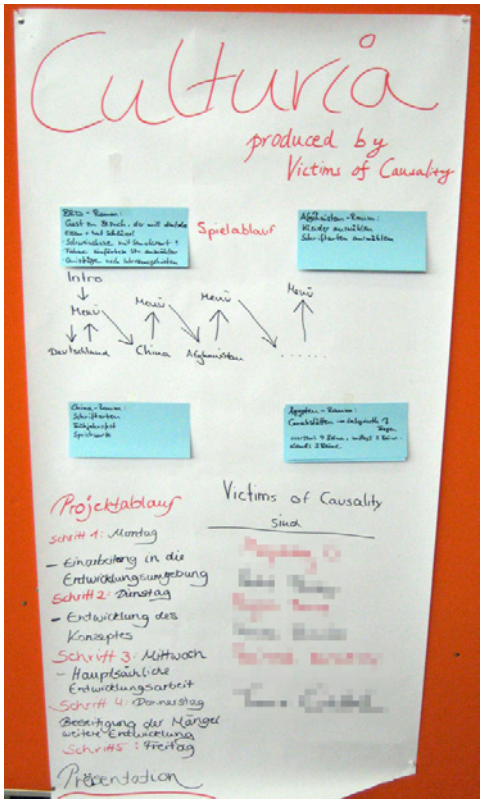



Abbildung 6.3: Das Poster der Projektwoche am KiWi. Die Namen wurden nachträglich anonymisiert.

Paar bildete (14 und 15 Jahre alt), in dem er vorrangig arbeitete. So wechselte er immer mal wieder das Paar und besprach sich mit dem Älteren, wie es weitergehen sollte, bevor er wieder aktiv im Paar mit dem Jüngeren programmierte. Dies ist zwar streng genommen keine ordentliche Umsetzung des *pair programming* nach den agilen Methoden, jedoch eine den Umständen geschuldete hinreichend gute Lösung. Man kann also von zwei vollwertigen Paaren sprechen, die jeweils einen Raum entwickelten. Bei der Paareinteilung achteten sie natürlich darauf, dass der jeweilige Migrationshintergrund eines Teilnehmers mit der Nationalität des zu entwerfenden Raums übereinstimmte. Das jüngste Paar (beide 12 Jahre alt) hatte deutlich Probleme, mit dem Tempo der anderen mitzuhalten. Jedoch gab es immer wieder Pausen, in denen ihnen ein Älterer zur Hilfe eilte. Aufgrund der Gruppengröße und der Raumkonstellation (alle saßen nebeneinander an einem langen Tisch) war es nämlich den Teilnehmern jederzeit möglich, den Fortschritt der anderen zu erfragen oder die anderen Gruppen darüber zu informieren, wenn programmiertechnische Probleme existierten. Ein weiteres Phänomen war, dass benötigtes Material (Bilder, Klänge, Musik) per USB-Speicherstift verteilt wurde, nachdem sie häufig an einem Computer

in einem anderen Raum, der über einen Internetanschluss verfügte, heruntergeladen worden waren. Dabei wurde jedoch während der gesamten Woche kein Verfahren etabliert, das gewährleistete, dass alle Teilnehmer auf dieselbe eigene angelegte Mediendatenbank zugreifen konnten. In vielen Fällen kam es auch zu dem Problem, dass bestimmte Medien wieder gelöscht wurden, um Platz für neue zu schaffen, ohne dass diese vermeintlich veralteten Objekte vorher auf allen Computern abgesichert wurden.

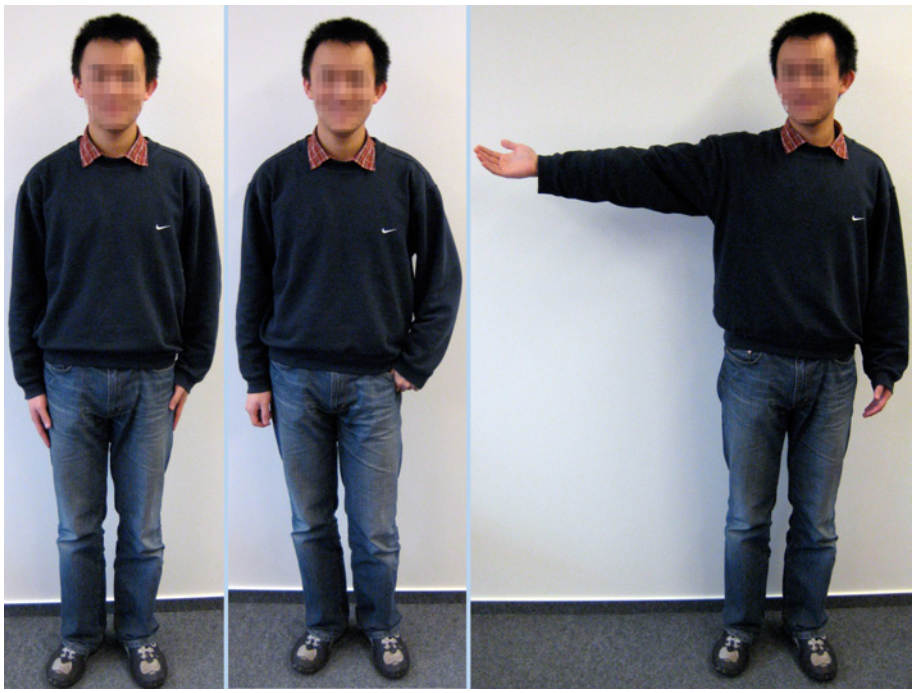


Abbildung 6.4: Bilder eines Teilnehmers der Projektwoche am KiWi zur Einbettung in das Spiel. Die Aufnahmen wurden nachträglich anonymisiert.

Die beiden älteren Teilnehmer waren begeistert von der Art des Projekts, der Thematik und dem ILE *scratch*. Sie engagierten sich innerhalb der Gruppe dafür, dass das entwickelte Spiel auch nach der Projektwoche weiter verfeinert wurde, neue Räume hinzukamen und eine Skriptübersicht angefertigt wurde (siehe Abbildung 6.5). So war es ihnen möglich, einen weiteren türkischen Raum mit einem ihrer Lehrer mit türkischem Migrationshintergrund als Charakter zu entwickeln, auch wenn dabei immer mehr der jüngeren Teilnehmer aus dem Projekt ausstiegen. Sie bewarben sich mit dem entstandenen Spiel außerdem bei einem Ideenwettbewerb der Schule, jedoch nur mit geringem Erfolg. Aus eigenem Antrieb organisierten sie ein weiteres Treffen nach der Projektwoche zu dem Spiel und präsentierten dort stolz ihre Fortschritte. Darüber hinaus besuchten diese beiden Teilnehmer im Folgejahr Pro-

jekte beim Schnupperstudium am Fachbereich Informatik der Universität Hamburg. Mehrmals versicherten sie, nach dem Abitur sich in einem Informatik-Studiengang einzuschreiben.

:
**Start**:start
**Beende**n:END
**JA**:keine
**Nein**:keine
**Title**:keine
**Version**:keine
**Wizard**:Fahne_erscheint,Peter_versteckt,chinesische_flagge_erscheint,af_fahne_erscheine2
credits,return
**D_Fahne**:fahne_deutsch_angeklickt,af_fahne_erscheine2
**Hamburg**:Hamburg,Antwort_Falsch
**Berlin**:Berlin,Antwort_Falsch

**Bonn**:bonn
**Bremen**:Bremen_Antwort_Falsch
**Sauerkraut**:Wurst_danach,deutsches_essen_fertig,sauerkraut_zuerst
**Wurst**:wurst_zuerst,sauerkraut_danach,deutsches_essen_fertig,WUrst_zuerst
**Döner**:döner,antwort_falsch
**Knödel**:Knödel,Antwort_falsch
**Schlüssel**:key_china_erscheint,key_china_verschwindet
**Andre**:Gerichte_erscheinen,frage_deutsche_hauptstadt,richtige_antwort,schlüssel_deutsch_er
scheint,schlüssel_deutsch_verschwindet
**Master_M**:NEW_room
**quizmaster_M**:question,china_durchegespielt

**Geld**: Keine
**Sprite1**: Scene, dialog1, dialog3, Antwort_Falsch, answer, Antwort_Falsch, Antwort_Falsch,
Antwort_Falsch
**Sprite2**:Tuzu
**C_Fahne:C**_Fahne_angeklickt, china_durchgespielt
**Sprite3**:kleider_gelöst
**Sprite9**:Antwort_Falsch
**Sprite8**:Keine
**AF_Fahne**:AF_Fahne_clickt
**Sprite10**:schrift_gelöst

| **Bühne** | return,spieler_verloren,frage_auswählen,bonuslevel_erreicht |
|---|---|
| | |
| | |
| | |
| | |
| | |

**Sprite7**:antwort_Falsch

Abbildung 6.5: Die manuell durch die beiden ältesten Teilnehmer angefertigte Skriptübersicht mit internen Skriptabhängigkeiten des Ergebnisses der Projektwoche am KiWi.

Die Beteiligung der Teilnehmer im CommSy-Raum war rege. Es ist dabei aber zu berücksichtigen, dass dieses Projekt über die eigentliche Projektwoche hinaus weitergeführt wurde und so der CommSy-Raum über die Zeit wachsen und sein Potenzial nachhaltiger entfalten konnte. Es existierten am Ende im CommSy-Raum 20 Ankündigungen, die zumeist von Teilnehmern kommentiert wurden. Den 15 verschiedenen Diskussionsthemen wurden durchschnittlich vier Beiträge durch die Teilnehmer hinzugefügt. Unter Materialien wurden mehrere Kategorien erstellt, die für die jeweiligen Versionsstände des *scratch*-Programms verwendet wurden. Auf die Fragen, was an *scratch* „doof“ bzw. „super“ war reagierten zwei Teilnehmer. Auch diese äußerten sich negativ über die Einfachheit und hoben gleichzeitig hervor, dass dies wiederum einen guten Einstieg ermöglicht. Die Frage, ob der interkulturelle Hintergrund „langweilig“ oder „spannend“ sei, wurde von vier Teilnehmern diskutiert. Sie sahen ebenfalls die gute Gelegenheit, mehr zu dieser Thematik zu lernen. Gleichzeitig mutmaßten sie jedoch, dass sich darin wenig spektakuläre Szenarien umsetzen lassen.

### 6.2.3 Projektwoche Haus der Jugend

Dieses Projekt wurde einer Gruppe von sechs Gesamtschülerinnen mit Migrationshintergrund angeboten, die gewöhnlich eine wöchentliche Hausaufgabenhilfe am Haus der Jugend (HDJ) in Hamburg-Wilhelmsburg wahrnahmen. Das Projekt wurde an drei aufeinanderfolgenden Tagen während der Frühjahrsferien 2009 in einem Computerraum des HDJ angeboten und die Teilnahme war freiwillig. Begleitet wurde der Termin von einem männlichen und einem weiblichen Betreuer, die auch sonst die Hausaufgabenhilfe anboten, beide hatten ein rudimentäres Verständnis von *scratch*. Die Betreuerin hatte ebenfalls einen türkischen Migrationshintergrund. Als Einführung von *scratch* wurde ein Aufgabenblatt verwendet, das Schritt für Schritt Anleitung gab und als Ergebnis ein *scratch*-Projekt lieferte, das zwei Charaktere miteinander kommunizieren und auf bestimmte Aktionen des Spielers reagieren lässt. Dieses Aufgabenblatt entstand während der ersten beiden Projekte, da sich herausstellte, dass eine schrittweise Anleitung über eine Präsentation häufig schwierig zu realisieren ist, da nur schwer auf unterschiedliche Arbeitsgeschwindigkeiten eingegangen werden kann. Dieses Aufgabenblatt wurde vor dem Einsatz mit Jugendlichen bei einem Workshop für Informatik-Lehrer eingesetzt (Das Aufgabenblatt befindet

sich im Anhang C). Zusätzlich standen den Teilnehmerinnen die *scratch*-Karten[5] zur Verfügung, die kleine gebräuchliche Funktionalitäten vorstellen und mit einem *scratch*-Skript lösen. Auf den zugehörigen CommSy-Raum und das Internet konnte während der gesamten Projektzeit von den Arbeitsrechnern zugegriffen werden.

### Ablauf

Der erste Tag des Projekts wurde dazu verwendet, sich mit *scratch* und dem CommSy vertraut zu machen. Nach Erläuterungen zum Vorgehen mit *pair programming* und dem täglichen *meeting* hatten die Teilnehmerinnen noch Zeit, erste Ideen für Geschichten mit interkulturellem Kontext zu entwickeln. Am zweiten Tag sollten sie diese Geschichten verfeinern und mittels *scratch* umsetzen. Am letzten Tag sollte die Umsetzung beendet sein und eine vorführbare Geschichte präsentiert werden.

| Zeiteinteilung | Tag 1 (180 min) |
|---|---|
| Begrüßung/Kennenlernen | 15 min |
| Einführung CommSy | 10 min |
| Aufgabenblatt *scratch* | 45 min |
| Einführung agile Methoden | 10 min |
| Brainstorming/Entwurf Geschichten | 100 min |
| Zeiteinteilung | Tag 2/3 (je 180 min) |
| freie Projektarbeit | je 180 min |

Tabelle 6.4: Der Ablaufplan des Projekts bei der Projektwoche am HDJ 2009

### Beobachtungen

Die Teilnehmerinnen hatten keinerlei Einwände gegen das Programmieren im Paar, es fiel jedoch auf, dass sie selbst innerhalb der Paare nur gering lösungsorientiert kommunizierten. Nach einer Brainstorming-Phase mit wenigen eigenen Ideen, einigten sich die Paare schnell auf ein quizspielartiges Szenario. Danach dienten die Gespräche nur noch dazu, Äußerlichkeiten der Charaktere und Objekte zu besprechen. Bei programmiertechnischen Fragen wandten sie sich sofort an einen Betreuer. Ebenfalls war ein Fokus auf das Malprogramm festzustellen, das dazu diente, Farben und ähnliches anzupassen.

[5] http://info.scratch.mit.edu/support/scratch_cards bzw. wurde die deutsche Übersetzung verwendet: http://www.brandhofer.cc/?p=65, zuletzt besucht am 15. April 2012.

Einige Teilnehmerinnen probierten auch die Vorschläge der *scratch*-Karten aus, wendeten jedoch diese erlernten Funktionen nicht weiter an, sondern verloren sich zumeist darin, verschiedene Hintergründe beim jeweiligen Skript auszuprobieren oder im Internet nach passenderen Hintergründen zu suchen.

Den meisten Teilnehmerinnen fiel es schwer nachzuvollziehen, dass den einzelnen Objekten Anweisungen mittels der Bausteine gegeben werden können. Auch die Logik der Bausteine erschloss sich ihnen nicht; so war ihnen beispielsweise nicht ersichtlich, dass die Werte der einzelnen Bausteine manuell angepasst werden konnten. Statt dessen bildeten sie teilweise Ketten von gleichen Bausteinen, um ein gewünschtes Ergebnis über die Summe der Bausteine zu erlangen. Trotz der offensichtlichen Schwierigkeiten mit *scratch* wirkte der Computer anziehend auf alle Teilnehmerinnen, wenngleich auch erwähnt werden muss, dass nur noch ein Paar nach dem ersten Tag zu den beiden Folgetagen erschien. Dies führte der Betreuer des HDJ jedoch auf Schul-Referate zurück, die während der Schulferien zu erledigen waren.

Letztlich war bei dem verbleibenden Paar zu erkennen, dass sie ausschließlich daran interessiert waren, für das Quiz-Spiel Bilder prominenter Persönlichkeiten im Internet zu suchen. Die Spiellogik dahinter war ihnen unwichtig, und man konnte sie auch nicht davon überzeugen, daran selbst mitzuentwickeln. So ergab sich, dass die Umsetzung des einfachen Spielprinzips mehr oder minder von den Betreuern diktiert wurde und die beiden Teilnehmerinnen dafür verantwortlich waren, die Skripte gemäß Anweisungen zu erstellen, Antwortmöglichkeiten vorzugeben, Reaktionen bei Erfolg bzw. Misserfolg zu definieren und die gefundenen Bilder einzubinden. Beim Austausch der Bilder zeigte sich, dass es ihnen schwer fiel, diese über E-Mail untereinander zu verteilen; den angebotenen CommSy-Raum verwendeten sie nicht.

### 6.2.4 Projektwoche Niels-Stensen-Gymnasium

Bei diesem Projekt handelte es sich um ein Angebot während der Projektwoche 2009 am Niels-Stensen-Gymnasium (NSG) in Hamburg-Harburg an dem 12 Schüler teilnahmen (davon waren sechs weiblich). Für die Projektwoche gab es jedoch ein stark einschränkendes Rahmenprogramm und für jede Klassenstufe ein vorgegebenes Thema. Somit musste sich die eigentliche Projektphase auf zweimalig zwei Zeitstunden beschränken. Daher wurde hier lediglich auf *pair programming* Wert gelegt und auf die einführenden Vorträge zur Softwareentwicklung und agilen Methoden verzichtet. Auch auf die Einführung von *scratch* konnte verzichtet werden, da nahezu alle Teilnehmer bereits einige Projekte damit in einer Informatik-AG absolviert hat-

ten[6]. In der Informatik-AG wurde großer Wert darauf gelegt, die Infrastruktur des Computerraums und des zugehörigen Netzwerks zu vermitteln, so dass alle Teilnehmer beispielsweise mit der Idee eines gemeinsamen Tauschlaufwerks im Netzwerk vertraut waren und dieses dementsprechend zum Tausch von Medien verwendeten. Begleitet wurde das Projekt von dem Lehrer, der auch die Informatik-AG leitete. Das bewährte Grundthema mit interkulturellem Kontext wurde durch das von der Schule vorgegebene Thema *Fairtrade* erweitert. Auf das Internet und einen eigenen CommSy-Raum konnte während der Projektzeit zugegriffen werden.

### Ablauf

Im Gegensatz zur Projektwoche am KiWi war der Aufbau von der Schule vorgegeben: Die ersten drei Tage wurden dazu genutzt, das Thema inhaltlich zu bearbeiten, und die beiden letzten Tage sollten dazu aufgewendet werden, diese Inhalte anschaulich zu verpacken. Demnach wurde *scratch* in den letzten beiden Tagen eingesetzt, um die entwickelten Geschichten zu *Fairtrade* multimedial umzusetzen (siehe auch Tabelle 6.5). Die wenigen Teilnehmer ohne *scratch*-Vorerfahrungen bildeten Paare mit Teilnehmern, die die meisten Erfahrungen aufweisen konnten (laut Aussagen des Lehrers). Auf diesem Wege konnten die gesamten vier Zeitstunden über beide Tage dafür verwendet werden, bereits konzipierte (interaktive) multimediale Geschichten mit *scratch* zu entwickeln.

| Zeiteinteilung | Tag 1 (120 min) |
|---|---|
| Begrüßung/Paareinteilung | 10 min |
| freie Projektarbeit | 110 min |
| Zeiteinteilung | Tag 2 (120 min) |
| freie Projektarbeit | 120 min |

Tabelle 6.5: Der Ablaufplan des Projekts bei der Projektwoche am NSG 2009

### Beobachtungen

In dem genutzten Computerraum existierten mehr Computer als teilnehmende Schüler, was dazu führte, dass das Programmieren im Paar einigen männlichen Teilnehmern nicht sinnvoll erschien. Diese Schüler fanden im Laufe des Projekts auch immer

[6]Das bereits erwähnte *scratch*-Aufgabenblatt (siehe Anhang C) wurde optional angeboten und auch von einer Teilnehmerin nebenher bearbeitet.

wieder „gute“ Begründungen, warum nun doch ein zweiter Rechner eingesetzt werden musste (z.B. eine Bildsuche im Internet oder das Bearbeiten von Grafiken). Die Mädchen dagegen arbeiteten wie selbstverständlich ausschließlich in Paaren, und es war darüber hinaus offensichtlich, dass sie einen stärkeren Wert auf die Kommunikation ihres Vorhabens und der entsprechenden Handlungen legten als ihre männlichen Mitschüler.

Während alle Teilnehmer das Tauschlaufwerk kannten, hatten einige starke Probleme, die dort gefundenen Medien in *scratch* einzubinden; zumeist hatten sie Schwierigkeiten, im Dialogfenster „Datei öffnen“ die entsprechenden Ordner in der Hierarchie des Tauschlaufwerks zu finden. Darüber hinaus erschloss sich ihnen nicht, wann neue relevante Daten auf dem Tauschlaufwerk von anderen Teilnehmern hinterlegt wurden. Dieser Prozess musste zumeist vom Lehrer angeregt werden, indem er aktiv die Ergebnisse bzw. Medien der einzelnen Gruppen erwähnte und andere Paare animierte, sich diese über das Tauschlaufwerk zu besorgen. Über den relativ kurzen Zeitraum des Projekts wurde außerdem das Tauschlaufwerk bzw. dessen Ordnerstruktur immer chaotischer zum Abspeichern genutzt. Dies führte dazu, dass gegen Ende weder Autoren noch andere Teilnehmer abgelegte Daten einfach wieder finden konnten, da häufig der Speicherort nicht wieder zu finden und der genaue Name einer Datei nicht mehr bekannt war. Zusätzlich wurde im Laufe des Projekts ein privater USB-Speicherstift eingesetzt, um Daten auszutauschen, was dazu führte, dass verschiedene Versionen dort und auf dem Tauschlaufwerk existierten und einige Teilnehmer den Überblick verloren, welche Version nun zu nutzen sei. Der angebotene CommSy-Raum wurde gar nicht verwendet, da dieser nicht in die bestehenden Strukturen und Arbeitsprozesse zu passen schien.

### 6.2.5 Girls' Day Greenfoot

Das *Greenfoot*-Projekt wurde gemeinsam mit Dr. Axel Schmolitzky und zwei studentischen Hilfskräften 2009 am Fachbereich Informatik der Universität Hamburg durchgeführt. Aufgrund der kurzen Zeit erschien es sinnvoller, ein Spielszenario und Bausteine dafür vorzugeben. Den vierzehn teilnehmenden Schülerinnen wurde eine leicht nachzuvollziehende deutschsprachige API angeboten, die mittels Methoden erlaubte, Objekte über die Cursortasten zu bewegen, Drehungen am Objekt vorzunehmen und die Kollision mit anderen Objekten abzufragen.

Dieses Projekt mit *Greenfoot* und die zwei darauf folgend beschriebenen Projekte mit *scratch* waren aufgrund der geringen Zeit und der jungen Teilnehmerinnen

didaktisch-methodisch deutlich reduziert, hier kam jeweils nur *pair programming* zum Einsatz. Auf das Internet konnte jederzeit zugegriffen werden. Für die Girls' Days stand ein CommSy-Raum zur Verfügung, der jedoch hauptsächlich von Betreuern verwendet wurde.

### Ablauf

Dieses Projekt glich einem angeleiteten Kurs, in dem abwechselnd Programmierkonzepte und Funktionsweisen im Plenum vorgestellt und diese Konzepte dann direkt praktisch nachvollzogen wurden. Es handelte sich jeweils um kleinteilige Schritte, die nicht mehr als zwei Konzepte beinhalteten. Durch Beendigung des angeleiteten Teils entstand ein Ausgangsszenario, das es erlaubte, Objekte per Tastensteuerung durch einen zweidimensionalen Raum zu bewegen, welcher mit beweglichen und unbeweglichen Hindernissen angereichert war. Während der einzelnen Arbeitsschritte und nach dem angeleiteten Teil blieb den Teilnehmerinnen eine halbe Stunde Zeit, eigene Vorstellungen zu realisieren (z.B. das Hinzufügen eines zweiten Spielers, das Zählen von Punkten oder auch die Verwendung eigener Sprites für die Objekte).

### Beobachtungen

Auch hier war wieder zu festzustellen, dass *pair programming* durch die Schülerinnen als positiv wahrgenommen wird und keinerlei negative Aussagen oder Versuche, alleine zu arbeiten, hervorbrachten, obwohl auch hier wieder prinzipiell mehr Computer als Teilnehmer vorhanden waren. Das Verhalten im Paar kann man als nahezu vorbildlich beschreiben: Die Paare unterhielten sich rege und fast ausschließlich über ihre Vorhaben bezüglich der Entwicklung eines Spiels, sie stimmten sich untereinander mehrmals ab und einigten sich sogar darauf, abwechselnd die Tastatur zu verwenden (ohne konkrete Hinweise darauf durch die Betreuer).

Über die Paare hinweg wurden keine Versuche unternommen, Wissen oder auch Medien auszutauschen. Gerade bei Syntaxproblemen wurde schnell ein Betreuer um Hilfe gebeten. Häufig ging es dabei nicht um die Vermittlung des Konzepts, wie ein Syntaxfehler zu erkennen und zu beheben ist, sondern darum, im konkreten Fall die richtige Syntax genannt zu bekommen und diese anzuwenden. Eine mehrfach gehörte Aussage hierzu war: „Die Klammern nerven uns“. Alle Paare schafften es, in der Projektzeit einen spielbaren Prototyp zu entwickeln.

Einige Teilnehmerinnen störten sich an den durch die API vorgegebenen Objekten (z.B. ein *einsammelbarer Fisch*). An dieser Stelle wird deutlich, dass sie gerne

mehr kreative Freiräume gehabt hätten und eigentlich nach ihren eigenen Interessen Klassen gestalten wollten. Ihnen war dabei jedoch sicherlich nicht bewusst, dass dies in der *Greenfoot*-Umgebung einen höheren und komplexeren Programmieranteil bedeutet hätte.

### 6.2.6 Girls' Days scratch

Bei den angebotenen Projekten während der Girls' Days 2010 und 2011 am Fachbereich Informatik der Universität Hamburg wurde *scratch* verwendet. Die Projekte wurden gemeinsam mit Susanne Germer und zwei weiblichen studentischen Hilfskräften durchgeführt. Auch hier wurde aufgrund des Zeitmangels auf die interkulturelle Thematik und eine offene Brainstormingphase verzichtet. Statt dessen konnten die jeweils zwölf Teilnehmerinnen aus Karteikarten wählen, die jeweils Ausgangssituationen für kurze Geschichten mit beliebten Schauspielerinnen und Schauspielern vorschlugen. Die Verwendung der *scratch*-Karten wurde durch die Betreuer empfohlen, war jedoch optional. Teilnehmerinnen, die sich bereits vorher kannten, durften sich in Paaren zusammenfinden. Auf diesem Weg war es selten nötig, Paare aktiv einzuteilen.

#### Ablauf

Nach einer ca. zehnminütigen Einführung in die Funktionalitäten und Möglichkeiten von *scratch* wurden die einzelnen Themenvorschläge kurz präsentiert und darauf hingewiesen, dass die Auswahl frei sei, aber auch eigene Ideen umgesetzt werden können, wenn man sich dies in der knappen Zeit zutraut. Danach begann die freie Projektphase in den Paaren, wobei alle Betreuer zur Verfügung standen, um jederzeit auf Fragen zu reagieren.

#### Beobachtungen

Während alle Paare sich beim ersten Durchlauf für ein eigenes Thema entschieden hatten, waren sie beim zweiten Durchgang damit beschäftigt, eines der vorgegebenen Themen zu bearbeiten.

Alle Teilnehmerinnen beider Girls' Days arbeiteten mit großer Freude in Paaren. Sie diskutierten und lachten viel innerhalb der Paare. Es war auffällig, dass sie zwar interessiert waren an den Ergebnissen der anderen Paare, jedoch keine der Teilnehmerinnen auf die Idee kam, bei Problemen andere Paare zu befragen, ob sie vielleicht Lösungen kannten. Es war aber auch zu beobachten, dass sich „lustige“

Funktionen von *scratch*, die unweigerlich im ganzen Raum für alle Teilnehmer erfahrbar waren (Audioaufnahme und Fotoschnappschüsse), schnell über die Paare hinweg verbreiteten. Dies führte jedoch bei einigen wenigen dazu, sich nur noch an Details aufzuhalten und das Gleiche immer wieder zu überarbeiten. Bei programmiertechnischen Fragen wurden zumeist die Betreuer um Hilfe gebeten, recht häufig war in diesem Zuge die substantielle Frage zu hören, ob das Ziel überhaupt umsetzbar sei. Daran war deutlich zu erkennen, dass die Teilnehmerinnen sich über den Umfang und das Potenzial von *scratch* nicht im Klaren waren.

Beim ersten Durchlauf erwies sich, dass die kurze Einführung in *scratch* vollkommen ausreicht, um solch ein kleines Projekt durchführen zu können. In Teilen ist dies auch darauf zurückführen, dass *scratch* noch viele weitere Funktionen bietet, die dazu einladen etwas auszuprobieren, ohne konkret Skripte erstellen zu müssen (z.B. die Malanwendung oder die Auswahl von passenden Tonsamples). Auch war zu beobachten, dass im Rahmen eines Tagesprojektes trotz genügend Zeit Lösungswege kaum hinterfragt oder verfeinert wurden.

Beim zweiten Durchlauf gab es ein Paar, das eine minimale Lösung bereits nach der Hälfte der Zeit fertigstellte und danach sich nicht mehr auf ein gemeinsames Ziel einigen konnte. Die Mädchen durfte dann individuelle Projektideen mit *scratch* an jeweils eigenen Rechnern umsetzen. Dieses Paar war sowohl vom Alter als auch von den Erwartungen an den Girls' Day sehr unterschiedlich.

### 6.2.7 Begleittermine Informatik AG

Die Begleittermine bei der Informatik AG ergaben sich aus der guten Zusammenarbeit mit dem Informatik-Lehrer des NSG während der Projektwoche am NSG. Die Teilnehmergruppe setzte sich aus den Teilnehmern der Projektwoche am NSG zusammen, wobei zwei männliche Teilnehmer hinzukamen, da zwei weibliche Teilnehmer der Projektwoche der Informatik AG fern blieben. Diese Termine waren gegenüber den oben genannten Projekten grundlegend anders konzipiert: Sie dienten dazu, die Schüler als Design Informanten bzw. sogar als Design Partner zu gewinnen, um so weitere Gestaltungshinweise für einen um soziale Aspekte erweiterten Papierprototyp von *scratch* zu entwickeln (vergleiche auch Janneck [2006] zum Einsatz von Papierprototypen im Schulkontext). Die Begleittermine waren jeweils in sich abgeschlossene Einheiten und hatten meist konkrete Ziele oder Fragen an die Teilnehmer. Pro Termin stand jeweils eine Doppelstunde (ca. 90 min.) zum eigentlichen AG-Zeitraum zur Verfügung. Die ersten beiden Termine fanden im Juni/Juli

2009 statt. Der dritte Termin fand ein gutes halbes Jahr später statt (März 2010). Es nahmen zwölf Schüler teil, davon waren vier weiblich. Ein CommSy-Raum wurde weiterhin angeboten.

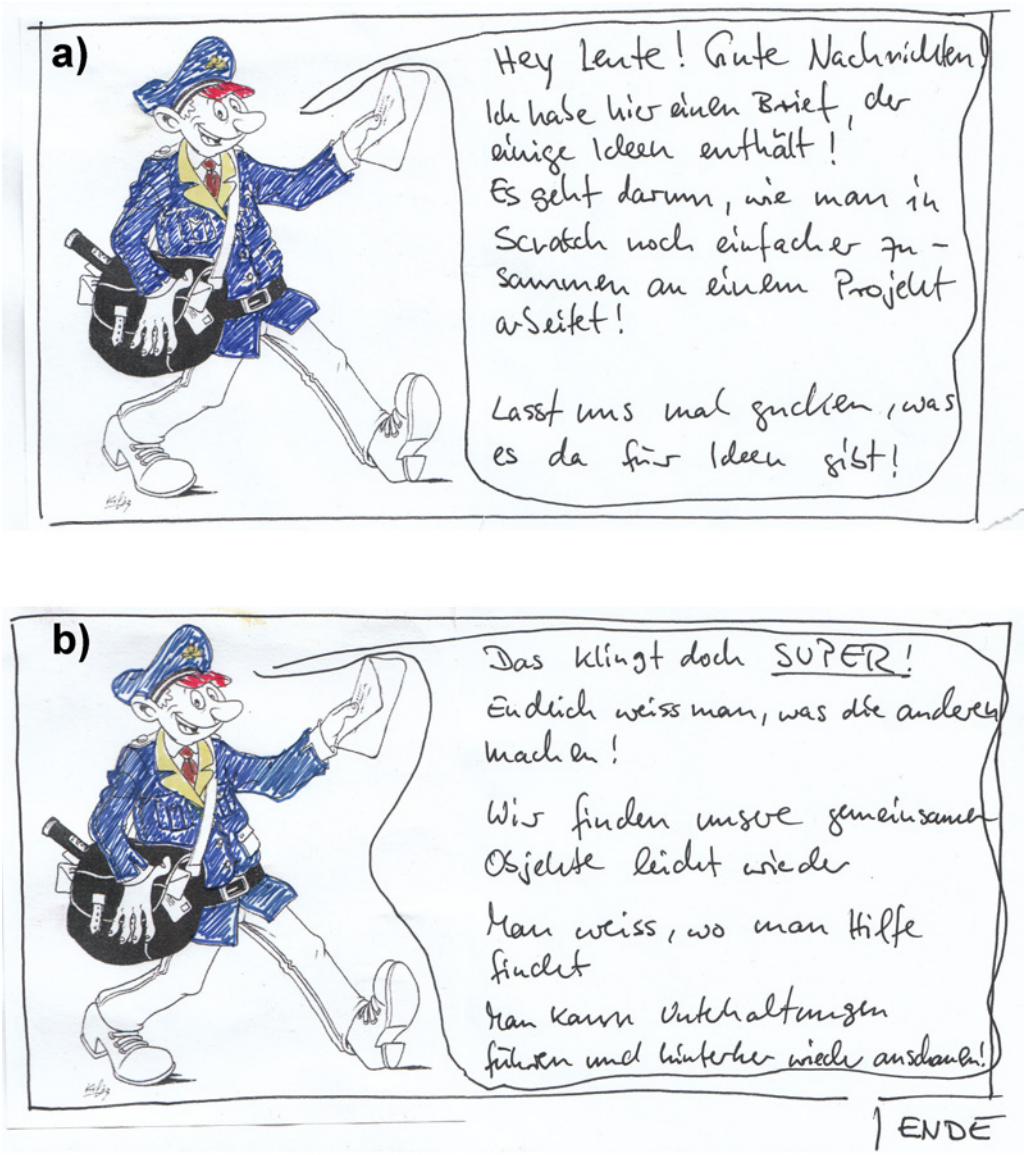



Abbildung 6.6: Das *Comicboarding* Ausgangsszenario (a) und das Endszenario (b) in der Papierversion. In der Variante mit *scratch* wurden zwei Skripte mit der gleichen Figur verwendet.

Der Auftakttermin hatte zur Aufgabe, den Teilnehmern in einer kleinen praktischen Einheit (ca. 40 min.) zu verdeutlichen, dass eine gemeinsame Gestaltung von Anwendungen in Kleingruppen mit *scratch* Probleme hervorrufen kann. Darauf folgend wurden sie für die verbleibende Zeit des Termins gebeten, mittels einer abgewandelten Version von *Comicboarding* (vergleiche Kapitel 4.1.1) Ideen zur Verbesserung von *scratch* zu generieren. Sie hatten dabei die Wahl zwischen einer Pa-

pierversion (siehe Abbildung 6.6) und einer *scratch*-Version (siehe Abbildung 6.7). Beide Versionen boten je ein Ausgangs- und ein zugehöriges Endszenario an, die Aufgabe der Teilnehmer war es, den mittleren Teil der Geschichte zu entwickeln und dabei eigene Ideen zur Erweiterung von *scratch* mit einzubauen. Diese Aufgaben wurden in Paaren bearbeitet.

Der zweite Termin fand in der Folgewoche statt und baute auf den Erfahrungen, welche die Teilnehmer im ersten Termin gesammelt hatten, auf. Hierbei ging es darum, einen Papierprototyp zu entwickeln, der die vorher erkannten Probleme und mögliche Lösungen in die *scratch*-Oberfläche einbindet. Dazu existierte eine Skizze von *scratch* auf einem DIN A3 Blatt (siehe Abbildung 6.8). Diese Skizze wurde von allen Teilnehmern gemeinsam (also nicht mehr in Paaren) mit Ideen angereichert und direkt als Funktionen in der *scratch*-Oberfläche angeordnet bzw. eingefügt.


Abbildung 6.7: Eine Szene des *Comicboarding* Ausgangsszenarios in *scratch*.

Im letzten Termin wurden kaum Bezüge zu den vorherigen Terminen hergestellt. Vielmehr lag der Fokus bei dieser Veranstaltung auf der Dokumentation von *scratch*-Programmen bzw. -Skripten und wie man diese so gestalten und dokumentieren könnte, dass sie auch von anderen Teilnehmern nachvollzogen und genutzt werden können. Dazu wurden die Teilnehmer gebeten, eine Lösung einer Aufgabe zu erarbeiten und diese dann so anzureichern, dass sie die entstandenen Skripte auch nach einer längeren Zeit (als Beispiel wurden die Sommerferien genannt) selbst noch nachvollziehen konnten. Es waren sämtliche Hilfsmittel erlaubt, z.B. das Verwenden der Kommentarblöcke in *scratch*, eines Textverarbeitungs- oder Präsentationspro-

gramms oder auch handschriftliche Notizen. Am Ende der Veranstaltung sollten alle Lösungen im Einzelgespräch vorgestellt werden.

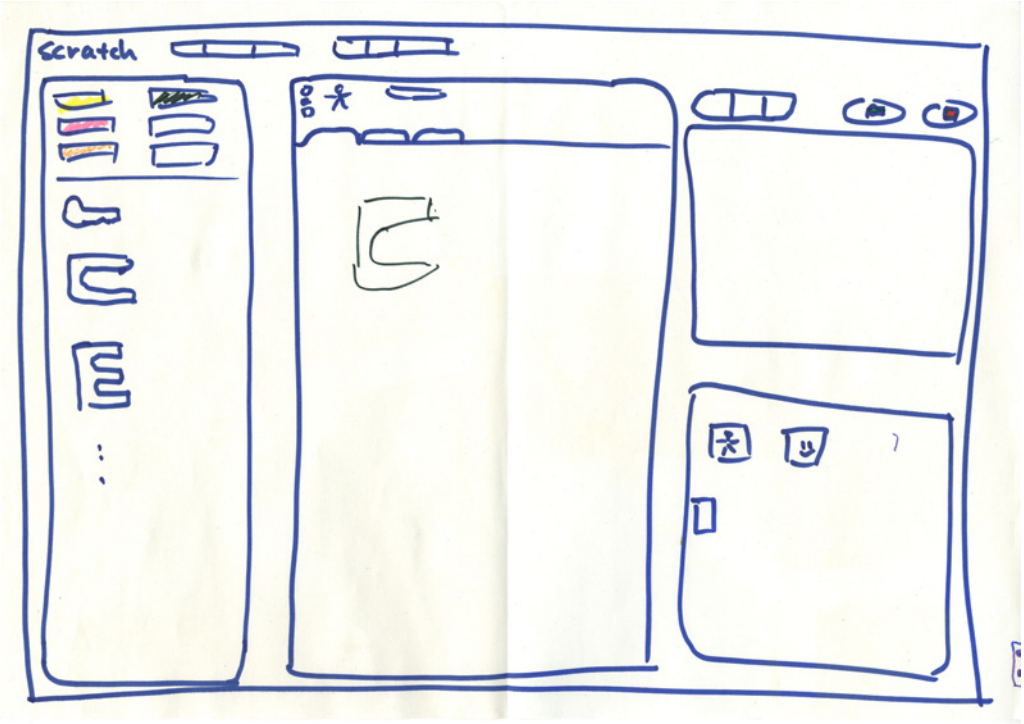



Abbildung 6.8: DIN A3 Skizze von *scratch*, die Schülern des NSG eine Grundlage für einen Papierprototyp zur Verfügung stellte.

### Beobachtungen

Im ersten Termin stellte sich heraus, dass die Schüler durchaus Verbesserungsbedarf in der Gruppenunterstützung von *scratch* sahen. Sie machten mit beiden *Comicboarding*-Varianten kleinere konkrete Vorschläge, beispielsweise fordern sie mehr grafische Objekte oder auch ein anpassbares Farbschema. Es war nicht festzustellen, dass eine der beiden Varianten bevorzugt wurde oder einen großen Einfluss auf die Ideenfindung der Teilnehmer hatte. Ein deutlich größerer Anteil wurde dazu verwendet, die Comiccharaktere und die Inhalte ansprechend zu gestalten. Der einzige Unterschied zwischen der Papiervariante und der *scratch*-Variante war darin zu konstatieren, dass bei *scratch* zusätzlich das Abstimmen von Animationen und Abläufen weitere Arbeitszeit in Anspruch nahm.

Beim Erstellen des Papierprototyps wurde klar, dass solch eine gemeinsame Methode bestimmte Personen stärker motiviert aktiv teilzunehmen als andere; so ergab sich, dass eine zielgerichtete Diskussion zwischen einer Teilnehmerin und einem

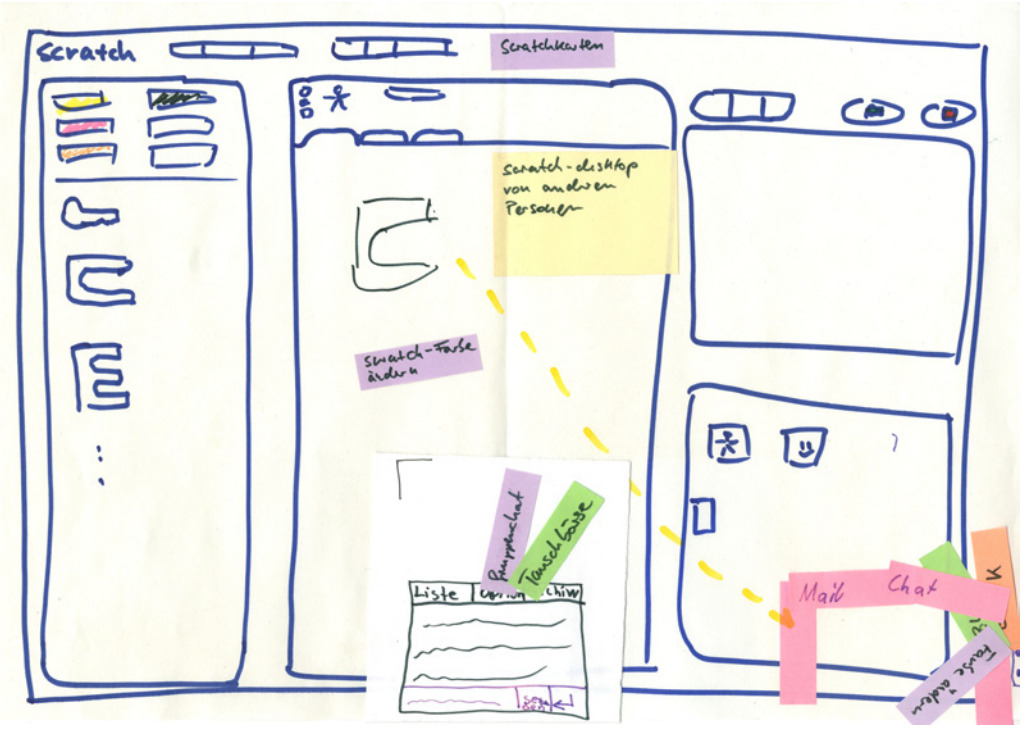



Abbildung 6.9: Der Papierprototyp, der gemeinsam mit Schülern des NSG als Erweiterung von *scratch* entwickelt wurde und eine Grundlage für den virtuellen Sandkasten darstellt.

Teilnehmer entstand, wohingegen die restlichen Teilnehmer zumeist nickend dabei standen. Es handelte sich bei dem Termin um den vorletzten Schultag vor den Sommerferien, so dass dieses passive Verhalten vielleicht auch ein wenig damit zu erklären ist. Die Diskussion förderte zu Tage, dass eine Funktion erwartet wurde, die es erlaubt, Gruppenmitgliedern Nachrichten zu senden oder ihnen Lösungen zukommen lassen zu können. Sie ordneten diese Funktionalität im unteren rechten Bereich von *scratch* an (siehe Abbildung 6.9).

Die Aufgaben des letzten Termins erschienen den meisten Teilnehmern zu abstrakt. Die Tatsache, dass man die eigenen Skripte zu einem späteren Zeitpunkt eventuell nicht mehr nachvollziehen kann, war den meisten fremd. Dies endete darin, dass beispielsweise zwei Teilnehmer den Sinn der Aufgabe nicht nachvollziehen konnten bzw. nur banale, nicht zu verwendende Lösungen präsentierten („Ich präge mir das ein und erkläre das in zwei Wochen“). Bei allen Teilnehmern gab es Probleme, die Funktion der Dokumentation überhaupt nachzuvollziehen, was zur Folge hatte, dass die meisten nicht die Skripte und Abläufe des Programms dokumentierten bzw. erklärten. Vielmehr bezogen sie sich zumeist auf die eigentlichen Funktionsweisen des fertigen Programms, so dass eher von einer Gebrauchsanweisung als einer Dokumentation die Rede sein kann. Es muss auch noch erwähnt werden, dass eigentlich

alle Teilnehmer das Onlinespiel *Plants vs. Zombies* begeistert nebenher spielten bzw. sich darüber austauschten und es so nur schwer zu erkennen war, ob die eigentliche Aufgabe vielleicht wegen dieser Gruppendynamik als zu trivial angesehen wurde.

### 6.2.8 Allgemeine Beobachtungen und Fazit

In allen Projekten konnte beobachtet werden, dass Hilfe-Foren von *scratch* bzw. *Greenfoot* aufgesucht wurden, diese jedoch nicht weiter genutzt wurden, da dort vorwiegend in Englisch kommuniziert wird. Dies schien die Schüler zu überfordern und häufig war ihnen in Folge auch nicht klar, wie sie ihre Fragen in Englisch formulieren bzw. welche Suchworte sie verwenden könnten.

Bei allen Projekten mit *scratch* war zu beobachten, dass der „Extras“-Knopf nicht verwendet oder gefunden wurde und somit die eigentlich vorhandene Möglichkeit, Skripte zu exportieren und zu importieren bzw. *scratch*-Projekte zusammenzuführen nicht genutzt wurde.

Zu den Projektbeobachtungen kann man zusammenfassend feststellen:

- Ein CommSy-Raum und auch ein *informative workspace* benötigen beim gleichzeitigen Einsatz von ILEs deutlich mehr gelenkte Aufmerksamkeit, um sie als Unterstützung für Gruppen zu vermitteln.
- Schüler benötigen außerdem Hilfestellungen beim Speichern/Verteilen von Medien und der Kommunikation.
- Unterschiedliche Herangehensweisen, Arbeitsentwürfe und Motivationen sind in Projekten zu berücksichtigen und aufzufangen.
- Auch auf verschiedene Arbeitsgeschwindigkeiten muss innerhalb der Gruppen hingewiesen werden.
- Den Teilnehmern muss klar sein, dass Hilfe auch innerhalb der Gruppe verfügbar ist und andere Teilnehmer gerne Unterstützung anbieten.

Auf diese Erkenntnisse wird im folgenden Kapitel nochmals eingegangen.

## 6.3 Kapitelzusammenfassung

In diesem Kapitel wurden die Eindrücke, dass soziale Aspekte nicht ausreichend in der schulischen Bildung und/oder in aktuellen ILEs unterstützt werden (Kapitel 3,

4 und 5), praktisch untersucht. In acht thematisch ähnlichen, aber strukturell unterschiedlichen Projekten mit insgesamt 75 Schülern im Alter von 10 bis 17 Jahren konnte beobachtet werden, dass in der Tat die Softwareentwicklung mit den gegebenen Werkzeugen, Methoden und Literaturhinweisen nur schwer als gemeinsame soziale Erfahrung zu vermitteln ist, da häufig notwendige Gruppenprozesse zu wenig unterstützt werden und zu selten gemeinsam auf das vorhandene Wissen der einzelnen Gruppenmitglieder zurückgegriffen wird. Die Beobachtungen zu den Problemen werden im folgenden Kapitel verwendet, um Anforderungen abzuleiten, die dabei helfen, Erweiterungen für ILEs mit dem Fokus auf soziale Aspekte zu entwickeln.

# Kapitel 7

# Schlussfolgerungen aus den Projektbeobachtungen

Aus den beschriebenen Beobachtungen zu den Projekten mit Schülern aus Kapitel 6 werden im Folgenden Anforderungen abgeleitet, um Jugendlichen im Schulkontext leichter zu ermöglichen, Gruppenprozesse in der Softwareentwicklung zu erkennen und zu nutzen.

Vorab sei noch einmal darauf hingewiesen, dass es sich nach den Überlegungen aus Abschnitt 4.1.2 bewusst um heterogene Projektkontexte und Teilnehmer handelte. Die Schüler wurden größtenteils als Informanten angesehen (vergleiche Kapitel 4.1.1), nur bei den Begleitterminen zur Informatik AG am NSG konnte man sie als Designpartner bezeichnen. Viele Herangehensweisen und Probleme, als Team zu agieren, waren trotz der Heterogenität in den meisten Projekten wieder zu entdecken. Dies spricht dafür, dass es sich nicht ausschließlich um eine Betrachtung für eine ganz spezielle Projektform handelt, sondern vielmehr davon ausgegangen werden kann, dass es sich hierbei um allgemeine Probleme von Gruppengemeinschaften mit gemeinsamen Zielen in der Benutzung der ILEs *scratch* und *Greenfoot* handelt (vergleiche Laughnan [2004] und Roussou et al. [2007], Kapitel 4.1.1). Diese sozialen Kontexte scheinen in beiden ILEs noch nicht ausreichend bedacht. Gleichwohl ist jedoch davon auszugehen, dass es noch weitere Probleme gibt, die nicht durch die präsentierten Projekte aufgedeckt werden konnten bzw. nicht explizites Ziel der Untersuchung waren.

In allen Projekten war zu erkennen, dass sich drei Phasen während der Entwicklungsprozesse in Gruppen herausbilden. Diese sind zu großen Teilen auf die Herangehensweisen der Schüler und deren Unterrichtsgewohnheiten zurückzuführen, können

aber in Teilen auch z.B. auf Theorien der computergestützten Kommunikation abgebildet werden, die von einem Lebenszyklus von Gruppen sprechen [Johansen et al., 1991]: Im ersten Teil, den Aufbauphasen, geht es darum, einen sozialen Kontext zu schaffen (*Orientierung* und *Vertrauen aufbauen*), eigene und gemeinsame Ziele und Rollen zu formulieren und verpflichtende Aufgaben zu verteilen. Im zweiten Teil, den Erhaltungsphasen, geht es vorrangig darum, Lösungen zu generieren und der Gruppe Informationen über das aktuelle Vorhaben zugänglich zu machen und gegebenenfalls Hilfe anzubieten. Im Schulkontext wird zudem von den Schülern meist eine weitere explizite Dokumentationsphase mit individuellen Ergebnissen erwartet. Diese Dokumentation wird sonst in der computergestützten Kommunikation nebenläufig erarbeitet. Diese Dokumentationsphase ist dazu gedacht, eine Arbeit zu verfassen, die dem Lehrer als (notenrelevantes) Endergebnis präsentiert werden kann und sowohl das Produkt als auch den Weg dorthin beschreibt. Durchgängig über sämtliche Phasen wird eine Infrastruktur benötigt, um gemeinsame Dokumente und Daten der Gruppe zur Verfügung stellen zu können. Die von Janneck und Janneck [2004] erwähnten Phasen mit auftauchenden Konflikten und einer entsprechenden Kompromissfindung war ebenfalls in den Projekten wieder zu entdecken. Diese bezogen sich meist auf programmiertechnische Umsetzungen, die sich für die Gruppen als praktikabel erwiesen.

Es war ebenfalls zu erkennen, dass Jugendliche nach Bezügen zur realen Softwareentwicklung suchen und diese gerne adaptieren. In diesem Zusammenhang ist festzustellen, dass weder auf die Literatur zur Didaktik der Informatik gestützt noch auf Erfahrungen von Lehrern zurückgreifend technische oder didaktische Hinweise herausgearbeitet werden können, welche Rollen einzunehmen sind und wie diese miteinander in Projekten interagieren.

Allgemein können anhand der Beobachtungen Anwendungsszenarien und damit verbundene Schwierigkeiten extrahiert werden (siehe Abbildung 7.1)[1], die die Aussage zulassen, dass es aktuellen ILEs wie *scratch* und *Greenfoot* derzeit noch an einer nötigen Fokussierung auf die Unterstützung sozialer Aspekte mangelt: Während diese ILEs hervorragend geeignet sind für das Arbeiten an einem Projekt alleine oder im Paar, tauchen die ersten Probleme auf, wenn zwei Nutzer an jeweils einem Rechner ein gemeinsames Projekt bearbeiten wollen. Die größte Schwierigkeit besteht dann jedoch darin, mehrere Paare zu unterstützen, die an einem gemeinsamen Projekt arbeiten. Dieses am schwierigsten umzusetzende Szenario entspricht jedoch am ehesten der aktuellen professionellen Softwareentwicklung und böte sich daher

[1] Diese Szenarien wurden auf der Posterdemonstration zu Göttel [2009b] ebenfalls präsentiert.

als geeignetes Mittel an, um die Attraktivität des Berufsfelds IT im Anfangsunterricht zu vermitteln. Dementsprechend sind technische und didaktisch-methodische Hilfestellungen für dieses Szenario zu entwerfen.

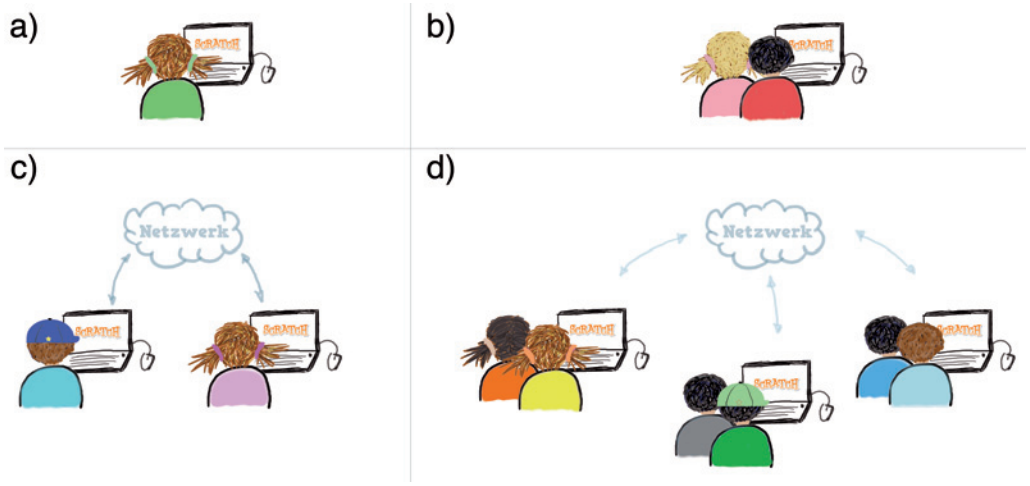



Abbildung 7.1: Vier mögliche Anwendungsszenarien für *scratch* im Schulkontext: Arbeit alleine (a), im Paar (b), jeweils alleine aber mit gemeinsamen Zielen und Ergebnis (c) und in Paaren mit gemeinsamen Zielen und Ergebnis (d).

## 7.1 Anforderungen zur Stärkung sozialer Aspekte der Informatik

Die nachfolgend genannten Anforderungen gehen auf die oben angesprochenen Probleme der Schüler während der Entwicklung eines gemeinsamen Produkts in Paaren bzw. Kleingruppen ein. Sämtliche Anforderungen zur Stärkung der sozialen Aspekte beinhalten demnach Punkte, die allesamt auf mangelnde soziale Unterstützung oder Gruppenkoordination zurückzuführen sind. Wenn vorhanden, werden weitere Aspekte der einzelnen Anforderungen, unter Anführung von Literaturquellen, präsentiert.

### 7.1.1 Bezugspunkt Gruppe

Die Beobachtungen zeigen, dass die Schüler fokussiert auf die Oberfläche von *scratch* waren. Weder nebenläufige Programme, wie etwa der Webbrowser[2] für das CommSy, noch externe Plattformen, wie der *informative workspace* wurden von sich aus so

[2]Einzige Ausnahme stellte ein Onlinespiel dar, das in einer Einheit der Begleittermine ausführlich beachtet wurde. Hierbei handelte es sich aber nicht um eine Programmieraufgabe in *scratch*.

ausreichend beachtet bzw. bearbeitet, als dass sie einen Mehrwert für die Gruppe darstellen konnten. Hier scheint es notwendig, die Aufmerksamkeit zu bestimmten Situationen, die der Gruppendynamik dienlich sind, dahin zu leiten, um für einen Anreiz zum Besuch der entsprechenden Plattform zu sorgen, möglichst innerhalb des Systems, das die Hauptaufmerksamkeit genießt.

Ein weiterer Erklärungsversuch könnte sein, dass die gleichzeitige Verwendung verschiedener mehr oder minder neuer/unbekannter Systeme besonders junge Schüler überfordert. Auch diese Überlegung führt zur Forderung, Schnittstellen zu Gruppenaktivitäten innerhalb des primär fokussierten Systems anzubieten und gleichzeitig etablierte Plattformen zu verwenden, um die Chance zu erhöhen, dass Schüler zumindest Grundzüge dieser Umgebungen bereits kennen.

Funktionen, die aus einem System heraus auf Ereignisse und Ergebnisse anderer Gruppenteilnehmer hinweisen, schaffen *awareness* und sind so in der Lage, das Gruppengefühl zu stärken bzw. ein solches aufkommen zu lassen (vergleiche Kapitel 3.1). Anzeigen, die auf Ereignisse hinweisen und die Möglichkeit zu späterem Aufsuchen geben, erlauben es den Teilnehmern, selbst zu entscheiden, wann sie sich dieser Meldung annehmen. Diese Form der Benachrichtigung ist gerade bei Jugendlichen empfehlenswert, da sie so lernen können, ihre Vorgehensweisen selbst zu organisieren und an Gruppenprozesse anzupassen. Dies stellt entsprechend der Forderungen von Fisker et al. [2008] in Kapitel 3.1 einen klaren Bezug zu gebräuchlichen Arbeitsformen in der IT-Branche her.

### 7.1.2 Kollektive Ressourcen

Die Verfügbarkeit der verwendeten Medien innerhalb der Projektgruppe ist möglichst einheitlich zu regeln. Wie die Verwirrungen in den Projektwochen am KiWi und am HDJ um das Abspeichern und das Verteilen der Dateien verdeutlichten, scheinen lokale und auch Netzlaufwerke keine praktikable Lösung zu sein. Von dieser Problematik wird beispielsweise aber auch in Literaturquellen berichtet [Schubert et al., 2011, S. 292 f.]. Vielversprechend ist hier, jeder Projektgruppe online einen Raum zur Verfügung zu stellen, auf den zu jeder Zeit und von überall zugegriffen werden kann. Nur Medien, die für dieses Projekt relevant sind, sollten zur Verfügung stehen. Auf eine Ordnerstruktur ist dabei zu verzichten, wie die Verwirrung um das Tauschlaufwerk in der Projektwoche am NSG verdeutlichte. Über diese grundlegende Funktion hinaus ist die Möglichkeit anzubieten, Kommentare zu hinterlassen, Fragen zu stellen und Diskussionen über die Ziele des Projekts zu

starten. So können auch die Forderungen von Henriksen et al. [2010] (Kapitel 4.2.2, S. 65) berücksichtigt werden, die besagen, dass bei Lernumgebungen besonders der Medienaustausch und der Ideenaustausch zu fördern ist. Vorteilhaft ist dabei auch die von O'Malley [1992] (Kapitel 4.2) geforderte Möglichkeit zur synchronen und asynchronen Kommunikation.

In allen Projekten war eine CommSy Infrastruktur vorhanden, die jedoch nur selten ausführlich genutzt wurde, da der Fokus hauptsächlich auf *scratch* und nicht wie bei Otto [2002] (Kapitel 4.2.1) ausschließlich auf das CommSy gelegt wurde. Entscheidend ist daher, auf eine solche Plattform permanent hinweisen zu können, um diese ins Bewusstsein zu bringen und Aktionen darin anzuregen.

Die Möglichkeit, Medien immer an einem bestimmten Ort im Projektkontext aufsuchen zu können, erlaubt einen strukturierteren Vorgang in der Entwicklung, aber auch eine weitere Verstärkung des Gruppencharakters, da in solch einem Raum die Teilnehmer namentlich ihre Ergebnisse zugänglich machen. Durch Kommentar- und Diskussionsfunktion werden automatisch Dynamiken der Gruppe sichtbar und dienen ebenfalls einem regen Austausch vor Ort, da Aussagen, die online getroffen wurden, häufig als Diskussionsgrundlage in der Präsenzzeit des Projekts dienen können.

### 7.1.3 Gemeinschaftsgefühl

Dem Drang einiger männlicher Teilnehmer zur Arbeit über Nacht –wie beispielsweise beim Schnupperstudium –ist nicht vollständig nachzugeben, man sollte jedoch im Hinterkopf behalten, dass solche Arbeiten, die über Nacht und zuhause angefertigt werden, nicht zu verhindern sind. Daher ist es ratsam, auch hier Möglichkeiten anzubieten, die diese Aktivitäten in einen gemeinschaftlichen Kontext einbinden. Hierbei sei besonders an *social media* Kommunikationskanäle gedacht, von denen anzunehmen ist, dass sie Jugendlichen geläufig sind, da es ausdrücklich nicht darum geht, diesen Teilnehmern noch zusätzliche Arbeitspakete zu schnüren.

Gerade durch Angebote, die die extern erledigten Arbeitsergebnisse wieder der Gruppe zugänglich machen, wird den Teilnehmern verdeutlicht, dass selbst einzelne Aktionen wieder als Bestandteil der Gruppe gesehen werden und in das gemeinsame Produkt einfließen. Die Möglichkeit, diese Ergebnisse zu verwenden und gegebenenfalls Autoren um Erklärungen zum Vorgehen zu bitten, erlaubt einen Umgang mit Programmcode fernab von der häufig anzutreffenden Idee unter (unsicheren) Teilnehmern, dass Software auf nicht nachvollziehbare Weise von unnahbaren Ent-

wicklern produziert wird. Das gemeinsame Nachvollziehen von Lösungen Einzelner kann so Gruppen verdeutlichen, was zu schaffen ist und wie man in der weiteren Entwicklung vorgehen kann.

### 7.1.4 Wissensaustausch

Es wurde deutlich, dass gerade soziale Aspekte wie Kommunikation mit anderen Teilnehmern oder konkreter das Eingestehen von Fehlern oder Problemen einigen Schülern schwer fallen, beispielsweise auch im Schnupperstudium zu erkennen. Andere Teilnehmer sind wiederum sehr kommunikativ, haben jedoch immense Probleme bei der programmiertechnischen Umsetzung ihrer Vorhaben. Dies war gut zu beobachten bei der Projektwoche am NSG und den Girls' Days (*scratch* und *Greenfoot*). In beiden Fällen macht sich somit eine Unsicherheit im Umgang mit Entwicklungsprozessen bemerkbar, die auch von Burnett et al. [2010] beschrieben wurde. Es gilt, diese Ängste und Unsicherheiten mit passenden technischen Unterstützungen und Unterrichtsformen zu nehmen. Auf lange Sicht ist hierbei entscheidend zu vermitteln, dass soziale Aspekte in der Softwareentwicklung üblich sind und folglich einen Gewinn darstellen. Die Kommunikation in den Projekten verlief immer dann erfolgreich, wenn sie geleitet wurde; daher ist es ratsam Methoden anzubieten, die die Teilnehmer ermutigen, sich mit der sozialen Gemeinschaft auszutauschen.

Auch wenn es im Schulkontext zumeist nur im kleinen Maße möglich ist, sind auch anonyme Austauschmöglichkeiten anzubieten. So gab es häufig den Fall, dass kleine Gruppen ihren Unmut über *scratch* laut kundtaten und so wahrscheinlich die Mehrheit der Teilnehmer einschüchterten, die eigentlich mit *scratch* zufrieden und von den komplexen Fähigkeiten überzeugt waren. Eine anonyme Möglichkeit, diesen negativen Aussagen entgegenzutreten, würde sicherlich die Arbeitsatmosphäre für die gesamte Gruppe stärken, da so eine Streitkultur entstünde, die die tatsächlichen Möglichkeiten von *scratch* herausarbeitet. Diese Problematik ist in der CKG (und der Psychologie) als ein Aspekt des *groupthink*-Phänomens bekannt. Dort werden ebenfalls anonyme Kanäle zu bestimmten Situationen empfohlen in denen Rollen- oder Hierarchiedenken hinderlich sein kann (vergleiche beispielsweise Bergstrom und Karahalios [2009]).

Andere Teilnehmer wiederum funktionierten als geschlossenes Paar so gut, dass sie jeglichen sozialen Kontakt zur Außenwelt vermieden. Es wäre nicht angebracht, solche Prozesse in Paaren komplett zu unterbinden, es sollten vielmehr weitere Mög-

lichkeiten angeboten werden, die einen Austausch mit anderen Teilnehmern über andere Medien erlauben.

Laut dem Konstruktionismus nach Papert [1980] ist es auch bei der Kommunikation sinnvoll, verschiedene Zugänge zu einer Gruppenkommunikation anzubieten, um so den Teilnehmern die Möglichkeit zu geben, die Form zu wählen oder auch selbst zu entwickeln, die sie am angenehmsten finden. Sind solche Wege gefunden, erkennen die meisten Teilnehmer sich als ein vollwertiges Mitglied der Gruppe, da ihre Stimme gehört wird und sie aktiv zu Ergebnissen beitragen können.

### 7.1.5 Kenntnis Projektfortschritt

Über die Paare hinweg kam es immer wieder zu unterschiedlichen Arbeitsgeschwindigkeiten, die meist zu spät der Gruppe bewusst wurden und so nur schwer abzufangen waren (Schnupperstudium und Projektwoche am KiWi). Daher müssen möglichst frühzeitig Wege gefunden werden, die verschiedenen Geschwindigkeiten zu veranschaulichen, um gegebenenfalls darauf eingehen zu können und zu verdeutlichen, dass langsame Paare entweder Unterstützung durch andere Teilnehmer erhalten müssen oder ihnen bereits erarbeitete Lösungen von wiederkehrenden Teilproblemen angeboten werden können.

Die aktive Hilfe durch andere Teilnehmer oder auch die Verwendung von deren Skripten ermöglicht eine weitere Stärkung des Wir-Gefühls bezüglich der erarbeiteten Lösung. Gerade auch die Möglichkeit anderen zu helfen bietet den Beteiligten die Gelegenheit, soziale Kompetenzen auszubauen und diese als Teil der Softwareentwicklung wahrzunehmen. Beispielsweise kommt es auch Caspersen und Kölling [2009] (Kapitel 3.1.1) darauf an, dass in der Informatikbildung auf die nötigen, aber auch quälenden und mühsamen Prozesse der Fehlerkorrektur eingegangen wird. Sie fordern, dass diese Abläufe als normal dargestellt werden, um nicht den Eindruck entstehen zu lassen, es gäbe Programmierer, bei denen solche langwierigen Probleme niemals auftauchen. Das Sichtbarmachen von Problemen verschiedener Teilnehmer und der konstruktive Umgang mit diesen Problemen durch andere Teilnehmer und nicht durch einen Lehrer oder Dozenten ermöglicht eine entspannte Herangehensweisen an die Problemlösung. So kann zusätzlich das Gemeinschaftsgefühl gestärkt werden.

### 7.1.6 Internes Tutorennetzwerk

Während häufig zu beobachten war, dass die fortgeschrittenen Teilnehmer gerne und auch qualitativ hochwertig zur Problemlösung anderer Teilnehmer beitrugen, war zu erkennen, dass viele Schüler – wahrscheinlich aus Gewohnheit – schnell Betreuer um Hilfe bitten. Zumeist musste beispielsweise in der Projektwoche am NSG und den Begleitterminen zur Informatik AG durch die Betreuer vermittelt werden, so dass die Hilfe innerhalb der Gruppe ablaufen konnte ohne eine konkrete Hilfestellung durch die Betreuer. Diese nötige Vermittlung ist nicht wünschenswert und ist zu einem großen Teil darauf zurückzuführen, dass den Teilnehmern nicht ausreichend bewusst gemacht wird, dass das nötige Wissen innerhalb der Gruppe vorhanden und abrufbar ist. Ziel sollte es also sein, dieses Wissen der einzelnen Teilnehmer problemorientiert zu veranschaulichen und die Möglichkeit einer direkten Kontaktaufnahme zu bieten. Es ist durchaus erstrebenswert, dass sich dabei auch *Spezialisten* für bestimmte Themen kennzeichnen lassen. Diese Kennzeichnung sollte idealerweise in der Hand der Teilnehmer liegen und nicht von Lehrern oder Betreuern festgelegt werden.

Nach den Erkenntnissen von O'Malley [1992] aus Kapitel 4.2 ist es empfehlenswert, diese Anforderung nicht ausschließlich auf technischer Ebene anzugehen, da solche Lösungen direkten Augenkontakt erschweren oder gar verhindern und das Gefühl für den selben Raum und die selbe Zeit nicht vermitteln können.

Während Betreuer bzw. Lehrer sicherlich auch Unterstützung und Hilfe anbieten sollten, ist es für die Stärkung des Gruppengefühls der Schüler eminent wichtig, dass eine Gruppendynamik entstehen kann, die den Teilnehmern das positive und motivierende Gefühl vermittelt, jedes Problem gemeinsam angehen und lösen zu können.

### 7.1.7 Programmvisualisierung

Darüber hinaus tauchte ein weiterer Aspekt auf, der jedoch nur für einige wenige Teilnehmer relevant zu sein scheint. So gab es z.B. die beiden älteren Personen aus der Projektwoche am NSG, die versucht haben, sich den zeitlichen Ablauf der Skripte zu visualisieren (siehe Abbildung 6.5). Diese Visualisierung fiel, wie in der Abbildung zu erkennen, sehr kryptisch aus. Den Teilnehmern wäre daher sicherlich geholfen, wenn es aus *scratch* heraus bereits die Möglichkeit einer externen und leicht verständlichen Visualisierung gäbe.

Visualisierungsmöglichkeiten mit Aussagekraft stellen gleichzeitig eine Diskussionsgrundlage für die gesamte Gruppe dar und können auch als Bestandteil der ge-

meinsamen Dokumentation verwendet werden. Ideal wäre dabei wieder der Zugang über die Kombination mehrerer Medien (z.B. Screenshots, Ausdrucke und/oder Aufzeichnungen).

### 7.1.8 Zusätzliche Anreize

Möglichkeiten, externe Personen auf Gruppenergebnisse hinweisen zu können, verliehen manchen Paaren und Gruppen einen zusätzlichen Motivationsschub. In der Folge konnten so sogar weitere Personen, die nicht der eigentlichen Projektgruppe zugehörig waren, mit einbezogen werden. Daher sollten neben der vorhandenen Möglichkeit zum Hochladen der Ergebnisse in *scratch* und *Greenfoot* weitere Angebote existieren, die es Jugendlichen erlauben, weitere Freunde und Bekannte über ihre Ergebnisse und Vorhaben zu unterrichten. Wenn möglich sollten auch Anreize geschaffen werden, sei es auch nur eine gemeinsame Präsentation vor einem externen Publikum.

Eine Plattform zu schaffen, die sämtliche Ergebnisse und Erkenntnisse einer Arbeitsgruppe externen Personen zugänglich macht, erlaubt es, neben einem Ergebnis auch auf das Gemeinsame daran hinzuweisen und so weitere Personen von der Idee zu überzeugen, Vorhaben mit informatischen Fragestellungen in Kleingruppen zu lösen. Gerade die Veranschaulichung von Kommunikation scheint die sozialen Aspekte der Softwareentwicklung gut widerzuspiegeln.

### 7.1.9 Vielfältigkeit

In seltenen Fällen kann es, wie beispielsweise im Schnupperstudium oder einem *scratch* Girls' Day, dazu kommen, dass einzelne Teilnehmer sich nicht in Paare oder die Gruppe integrieren lassen. Hier sollten Angebote zur Verfügung stehen, die es diesen Teilnehmern trotzdem erlauben etwas zu bearbeiten, wenn auch in Einzelarbeit. Auf diese Weise kann verhindert werden, dass ein gesamtes Gruppengefüge ins Wanken gerät. Das zu *scratch* zugehörige Malprogramm bietet sich dafür beispielsweise an. Mögliche Ergebnisse sind wiederum den anderen Teilnehmern zugänglich zu machen. So entsteht zumindest die theoretische Möglichkeit, die sich verweigernden Teilnehmer der Gruppe näher zu bringen.

Es ist in Erinnerung zu behalten, dass Personengruppen existieren, die keinen Zugang zum Programmieren finden und bei denen davon auszugehen ist, dass ihnen ein Werdegang im Berufsfeld IT nicht zu empfehlen ist. Solchen Gruppen sollten

Tätigkeiten angeboten werden, die es ihnen erlauben, andere Ergebnisse zu erarbeiten. Dabei muss ihnen jedoch klar gemacht werden, dass es sich dabei nicht um informatische Aspekte handelt, um keine falschen Hoffnungen entstehen zu lassen. Neben dem Malprogramm von *scratch* sollten möglichst zusätzliche Angebote zur Verfügung stehen, die zumindest kurzfristig den Gruppenfortschritt unterstützen.

## 7.2 Einordnung der Anforderungen

Aus den oben genannten Anforderungen zur Stärkung der sozialen Aspekte der Informatik wurden besonders relevante Bereiche anhand der auftretenden Häufigkeiten und den komplexen Folgen für die Gruppenarbeit festgestellt. Diese Bereiche lassen sich in drei Handlungsebenen zur Unterstützung und Stärkung sozialer Aspekte von Projektgruppen unterteilen[3]. Vergleicht man diese Ebenen mit den sozialen Aspekten, die nach Fisker et al. [2008] in ILEs integriert werden sollen, so kann man sagen, dass die erste Ebene sowohl als Austauschmöglichkeit für Artefakte als auch für Kommunikation steht. Die zweite Ebene stimmt mit der Forderung nach Unterstützung der *awareness* überein. Die dritte Ebene taucht bei Fisker et al. [2008] nicht auf und wurde hinzugefügt, um dem Schulkontext zu entsprechen, da dort benotbare Ergebnisse im Vordergrund stehen und nicht zwangsläufig funktionierende Softwareprodukte. Die Handlungsebenen sind wie folgt zu begreifen:

- *Austauschmöglichkeiten*: Gruppeninterne Möglichkeiten und Kanäle anbieten, um Ziele, Medien und Meinungen auszutauschen.
- *Awareness*: Bewusstsein für die Gruppe und die Aktivitäten der Teilnehmer schaffen und unterstützen.
- *Außendarstellung*: Externe Präsentation und Dokumentation von Gruppenergebnissen ermöglichen.

Diese drei Ebenen decken soziale Aspekte auf verschiedene Weisen ab. Während die ersten beiden Ebenen darauf abzielen, Gruppenprozesse in der Entwicklung zu veranschaulichen, zu erleichtern und zu betonen, zielt die dritte Ebene auf eine Stärkung des Gruppengefüges durch die Möglichkeit zur Außendarstellung ab. Diese externe Präsentation stellt zum einen einen motivierenden Faktor dar, und gibt zum

[3]Diese Ebenen wurden gemeinsam mit dem virtuellen Sandkasten (siehe Kapitel 8) auf der *scratch@MIT* Konferenz 2010 in Form einer Demonstration präsentiert (http://events.scratch.mit.edu/conference/index.php/Scratch/2010/announcement/view/9, zuletzt besucht am 15. April 2012).

anderen auch außenstehenden Betrachtern die Möglichkeit, die Softwareentwicklung als einen Gemeinschaftsprozess zu erkennen und zu begreifen. Tabelle 7.1 veranschaulicht die Zugehörigkeit der Anforderungen zur Stärkung der sozialen Aspekte zu den Ebenen, einige Anforderungen sind in mehreren Handlungsebenen zu verorten und erscheinen in der Tabelle somit mehrfach.

| | |
|---|---|
| Austauschmöglichkeiten | Kollektive Ressourcen<br>Gemeinschaftsgefühl<br>Kenntnis Projektfortschritt<br>Internes Tutorennetzwerk<br>Programmvisualisierung<br>Vielfältigkeit |
| *Awareness* | Bezugspunkt Gruppe<br>Kollektive Ressourcen<br>Wissensaustausch<br>Kenntnis Projektfortschritt |
| Außendarstellung | Programmvisualisierung<br>Zusätzliche Anreize |

Tabelle 7.1: Die ermittelten Anforderungen und ihre Verknüpfung mit den drei Handlungsebenen zur Stärkung der sozialen Aspekte.

Alle drei Ebenen sind durch Erweiterungen an *scratch* ansprechbar, es empfehlen sich jedoch in Anlehnung an das *blended learning* zusätzliche Anknüpfungspunkte auf einer didaktischen und persönlichen Ebene in Form von angepassten agilen Methoden. Nach wie vor wird in dieser Arbeit davon ausgegangen, dass mehrere Praktiken der agilen Methoden angeboten werden müssen, um soziale Aspekte der Softwareentwicklung genügend hervorzuheben. Im folgenden Kapitel werden sowohl die umgesetzten Erweiterungen für *scratch* als auch die agilen Methoden zum Einsatz im Schulkontext vorgestellt. Im Anschluss wird dann darauf eingegangen, wie diese prototypischen Erweiterungen und Methoden auf die hier gestellten Anforderungen zur Stärkung der sozialen Aspekte der Informatik wirken.

## 7.3 Kapitelzusammenfassung

In diesem Kapitel wurden Anforderungen aus den Beobachtungen zu den Projekten aus Kapitel 6 abgeleitet und mit Arbeiten aus Kapitel 3 in Beziehung gesetzt. Die relevantesten Anforderungen lassen sich in drei Handlungsebenen unterteilen (Austauschmöglichkeiten anbieten, *awareness* schaffen, Außendarstellung fördern), die

als Grundlage für eine Erweiterung von *scratch* dienten. Im folgenden Kapitel werden die Erweiterungen von *scratch* und nötige Anpassungen für die agilen Methoden im Schulkontext vorgestellt.

# Kapitel 8

# Der virtuelle Sandkasten und agile Methoden

Nach der Theorie des *blended learning* ist davon auszugehen, dass rein technische Lösungen Lernende nicht zufriedenstellend ansprechen und im Lernprozess unterstützen können. Es wird dabei davon ausgegangen, dass computergestützte Lernumgebungen nur funktionieren, wenn sie Hand in Hand mit methodisch-didaktischen Vorgehensweisen angewendet werden. Aus diesem Grund sind die hier präsentierten Lösungen zu den Anforderungen aus dem vorigen Kapitel sowohl technischer als auch methodisch-didaktischer Natur. Im ersten Teil wird eine Erweiterung von *scratch* namens der „virtuelle Sandkasten“ und im zweiten Teil werden Anpassungen von agilen Methoden an den Schulkontext vorgestellt. Die Erweiterungen und Anpassungen orientieren sich an etablierten Systemen und Praktiken der Softwareentwicklung, um einen Bezug zu realen Prozessen gewährleisten zu können.

Im ersten Teil werden folglich Fragen beantwortet, welche Erweiterungen der virtuelle Sandkasten anbietet und wie diese implementiert wurden. Der zweite Teil legt dar, durch welche Anpassungen es im Schulkontext gelingen kann, agile Methoden erfolgreich zu etablieren. Das Konzept des virtuellen Sandkastens lässt sich noch einmal gesondert bei Göttel [2009b] nachlesen, Beschreibungen zu agilen Methoden im Schulkontext finden sich bei Göttel [2011a]. Der virtuelle Sandkasten wurde darüber hinaus auf der scratch@MIT[1] Konferenz 2010 vorgestellt.

[1] http://events.scratch.mit.edu/conference/index.php/Scratch/2010/announcement/view/9, zuletzt besucht am 15. April 2012.

## 8.1 Der virtuelle Sandkasten

Beim virtuellen Sandkasten handelt es sich um einen Prototyp, mit welchem die am höchsten priorisierten Anforderungen aus Kapitel 7.1 auf technischer Ebene begegnet werden soll. Der virtuelle Sandkasten bietet in erster Linie Hilfen für die Erhaltungsphasen im Gruppenlebenszyklus und die Dokumentationsphase an (vergleiche Johansen et al. [1991]; Janneck und Janneck [2004] in Kapitel 3.2.1 und die allgemeinen Anforderungen aus Kapitel 7). Nach Bruckman [1998] (Kapitel 4.2.2) und der in den Projekten beobachteten starken Fokussierung von Schülern auf vorhandene Computer sollten technische Umgebungen den Nutzern Möglichkeiten anbieten, um miteinander in Kontakt zu treten, andere zum Mitmachen einzuladen und die Gruppe über Ergebnisse zu informieren. Henriksen et al. [2010] (ebenfalls Kapitel 4.2.2) empfehlen den Ideen- und Ressourcenaustausch in ILEs besonders zu fördern, um soziale Interaktion darin zu verdeutlichen und zu stützen.

Beim virtuellen Sandkasten handelt es sich um eine Erweiterung von *scratch* 1.3, die in einer gesonderten quelloffenen Version über eine offizielle Webseite[2] des *scratch*-Entwicklerteams heruntergeladen werden kann. *scratch* 1.x wurde in *Squeak*, einem Dialekt von *Smalltalk*, geschrieben, ist plattformunabhängig, verfügt über umfangreiche Bibliotheken und erlaubt so theoretisch, beliebige Änderungen an dem System vorzunehmen. Jedoch muss bedacht werden, dass es sich bei *Squeak* zwar um eine mächtige, jedoch auch etwas aus der Mode gekommene Entwicklungsumgebung handelt, so dass gerade aktuellere Technologien wie *social media* APIs nur umständlich einzubinden sind, da zumeist fehlerhafte/ungewartete oder nicht vorhandene Strukturen und Dokumentationen vorzufinden sind. Daher wurde während der Implementation eine Schnittstelle in *Python* realisiert, die mit *Squeak* und beispielsweise der *Twitter* API kommuniziert.

Der virtuelle Sandkasten greift alle drei Handlungsebenen aus Kapitel 7 auf, in dem die Beobachtungen gezeigt hatten, dass der Fokus bzw. die Aufmerksamkeit der Schüler zumeist auf eine Umgebung, nämlich *scratch*, eingeschränkt ist. Dies führte zu einem Dilemma, da es zum Einen nicht möglich und auch nicht sinnvoll ist, sämtliche Gruppenunterstützungen nativ in ein ILE einzubinden. Zum Anderen ist es jedoch fraglich, in Folge dessen auf den Funktionsumfang vorhandener Webplattformen gänzlich zu verzichten. Letztlich wurde Wert darauf gelegt, *scratch* in seiner klaren und übersichtlichen Form zu belassen und nur um leichtgewichtige Funktionalitäten zu erweitern, die es in bestimmten Situationen erlauben, den Fokus

[2] http://info.scratch.mit.edu/Source_Code, zuletzt besucht am 15. April 2012.

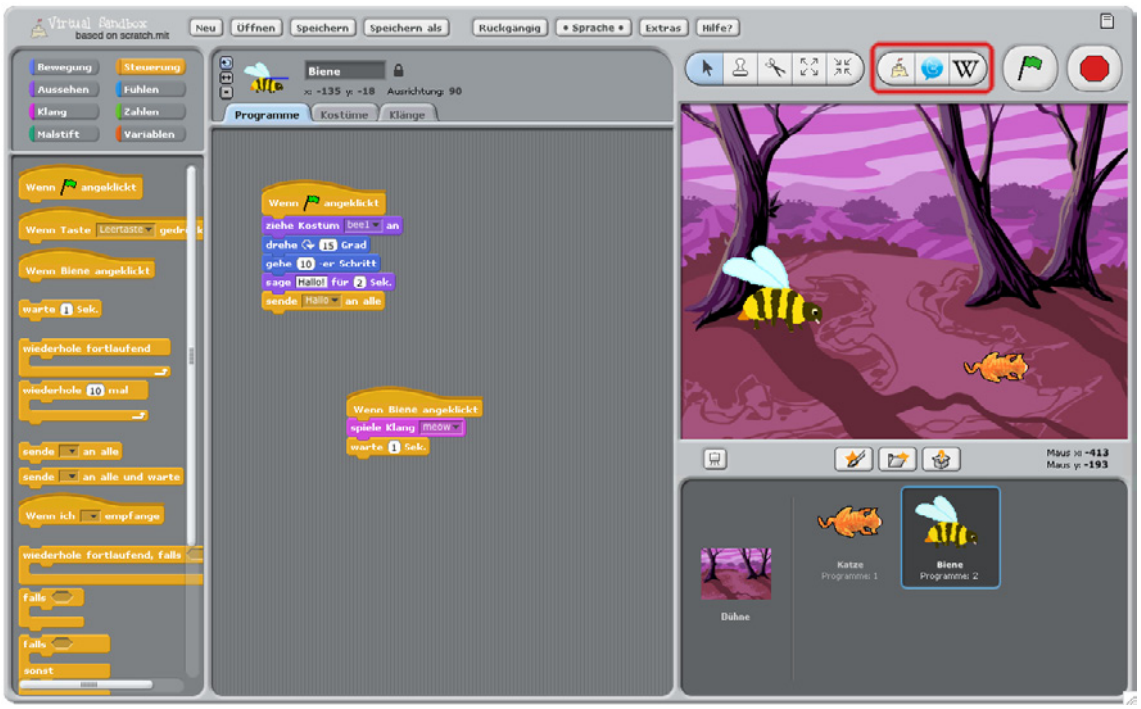



Abbildung 8.1: Die Oberfläche des virtuellen Sandkastens mit Erweiterungen (rot markiert) zur Gruppenunterstützung: „Gruppenplattform“, „Gruppenchat“ und „Wiki“. Der Knopf „Veröffentlichen!“ ist bei der quelloffenen Version nicht verfügbar.

auf eine andere Umgebung zu legen. So besteht die Hoffnung, dass vorhandene etablierte Gruppenunterstützungen, wie beispielsweise CommSy und Wikis (vergleiche Kapitel 3.2.1), besser wahrgenommen werden und auf diese Weise nach Bruckman [1998] und Henriksen et al. [2010] ein deutlicherer Fokus auf soziale Aspekte der Entwicklung gelegt wird. Eine Notwendigkeit, gezielt und dynamisch auf vorhandene Knöpfe innerhalb von *scratch* hinzuweisen, erscheint sinnvoll, da z.B. der vorhandene „Extras“-Knopf in keinem der Projekte von Teilnehmern verwendet wurde.

Beim virtuellen Sandkasten wurden drei offensichtliche Änderungen vorgenommen (siehe Abbildung 8.1, vergleiche mit Abbildung 5.2, S. 78): Es existieren nun drei weitere Knöpfe, die jeweils die identifizierten Anforderungsebenen abdecken sollen: Die Gruppenplattform, der Gruppenchat und das Wiki. Die Gruppenplattform wird als Onlineraum verstanden, der einer Gruppe ermöglicht, Ziele, Medien und Meinungen auszutauschen. Der Gruppenchat regt die Teilnehmer dazu an, in kurzen Mitteilungen aktuelle Ergebnisse oder auch Probleme mitzuteilen, um so ein Bewusstsein für die Gruppenaktivitäten zu schaffen. Das Wiki soll zur Außendarstellung der Gruppe und ihrer Ergebnisse genutzt werden. Um zu gewährleisten, dass diese drei Knöpfe eine hohe periphere Aufmerksamkeit bei den Nutzern erlangen,

wurde eine örtliche Nähe zu dem wichtigen und häufig verwendeten Knopf gewählt, der die Skripte bzw. ein *scratch*-Programm startet: Die Fahne. Diese örtliche Nähe zu einem häufigen Bezugspunkt ist bei Jugendlichen wichtiger als eine Einbettung in semantische Zusammenhänge.

Alle drei Knöpfe reagieren zeitnah auf Aktivitäten in den jeweiligen externen Plattformen. Sobald ein Eintrag in der Gruppenplattform, dem Gruppenchat oder dem Wiki ergänzt oder getätigt wurde, ändert sich das Aussehen der Knöpfe, um auf diese Neuigkeiten hinzuweisen (siehe Abbildung 8.2). Alle drei nachfolgend genannten, verwendeten Plattformen bieten RSS (*Really Simple Syndication*[3]) Dienste an.

Abbildung 8.2: Die Knöpfe Gruppenplattform, Gruppenchat und Wiki (von oben nach unten). Jeweils (von links nach rechts) in den Zuständen „Standard“, „Mouseover“, „Angeklickt“, „Benachrichtigung Standard“, „Benachrichtigung Mouseover“.

### 8.1.1 Die Gruppenplattform

Als System für die Gruppenplattform wurde das CommSy gewählt (siehe Kapitel 4.2.1), da die nötige Infrastruktur gut zugänglich war und Literaturquellen bereits vom erfolgreichen Einsatz des CommSys im Projektunterricht an Schulen berichten [Otto, 2002]. Ein Betätigen des Knopfs „Gruppenplattform“ öffnet ein neues Dialogfenster, das die letzten Neuigkeiten des jeweiligen CommSy-Projektraums chronologisch geordnet anzeigt. Von diesem Fenster kann man auch direkt zur Webplattform gelangen oder das Fenster wieder schließen (siehe Abbildung 8.3).

[3]Vormals *Rich Site Summary* oder auch *RDF Site Summary*. RDF wiederum steht für *Resource Description Framework*. RSS ist konzipiert, um Änderungen von Webseiten in einem standardisierten Format anzubieten und so eine Schnittstelle für entsprechende externe RSS-Applikation darzustellen.


Abbildung 8.3: Das Dialogfenster der Gruppenplattform. Aktuelle Aktivitäten des CommSy-Raums werden mit Autorennamen gelistet.

### 8.1.2 Der Gruppenchat

Wählt man den Knopf „Gruppenchat“ aus, so wird ein neues Dialogfenster angezeigt, das die Kommunikation aller Teilnehmer zeigt. Der Gruppenchat wurde mit der *social media* Anwendung *Twitter*[4] umgesetzt, so dass ein Gruppen-Tweet[5] existiert, dem alle registrierten Teilnehmer Nachrichten hinzufügen können. Im Dialogfenster des Gruppenchats kann man direkt Nachrichten versenden (siehe Abbildung 8.4), die dann mit einer beabsichtigten Zeitverzögerung im Gruppenchat veröffentlicht werden. Darüber hinaus ist nach dem Versenden der Nachrichten eine Zeitsperre für weiteres Versenden von 10 Sekunden realisiert, um der Dynamik und der damit häufig verbundenen vollständigen Eingebundenheit jugendlicher Teilnehmer in Echtzeitchats entgegenzuwirken. Der Gruppentweet ist immer auch über die entsprechende *Twitter*-Seite abrufbar, so dass er beispielsweise von zuhause angesehen und

[4]Twitter besitzt laut AGBs keine Altersbeschränkung und ist somit in einem Schulkontext einsetzbar.

[5]Ein Tweet bezeichnet die Abfolge aller Textmitteilungen, die von einer Person oder Gruppe über *Twitter* getätigt wurden.

zur Kommunikation genutzt werden kann. Hierzu bleibt zu sagen, dass es sich bei diesem Tweet grundsätzlich um einen nicht öffentlich zugänglichen Tweet handeln sollte, um die Privatsphäre der Jugendlichen zu wahren, den verantwortungsvollen Umgang mit Internettechnologien zu propagieren und sonstige Bedenken im Zusammenhang mit *social media* einzudämmen. In diesem Zusammenhang ist es mit Sicherheit sinnvoll, diese Thematik didaktisch aufzugreifen und mit Schülern zu diskutieren.


Abbildung 8.4: Das Dialogfenster des Gruppenchats. Im unteren Feld können Nachrichten versandt werden. Der Tweet erscheint im oberen Feld und ist auch über *Twitter* abrufbar.

### 8.1.3 Das Wiki

Der Wiki-Knopf stellt die einfachste Variante dieser drei Funktionalitäten getreu der Philosophie der Direktheit von Wikis dar. Hier wird der Nutzer über Änderungen am Wiki informiert und hat daraufhin die Möglichkeit, dieses Wiki direkt aufzusuchen, um die Änderungen zu verfolgen oder zu editieren. Da davon, wenn auch nicht gewünscht, auszugehen war, dass die dort geleistete Arbeit gegen Ende eines Projekts anfällt, erscheint eine Überleitung in eine andere Programmumgebung (Webbrowser) weg von *scratch* angebracht. Als Wikisystem kommt *Mediawiki*[6] zum Einsatz, da es sich hierbei um eine verbreitete Plattform mit Alltagsrelevanz handelt[7].

### 8.1.4 Das Kontextmenü

Weitere Änderungen wurden am Kontextmenü für Skripte vorgenommen, welches über die rechte Maustaste aufrufbar ist. Hier gibt es nun die Möglichkeit, sich die Historie der Autoren zu jedem Skript ausgeben zu lassen. Zusätzlich wird immer gleich im Kontextmenü der aktuell letzte Autor angezeigt. Besitzt ein Skript noch keine Historie, werden beide Möglichkeiten im Kontextmenü ausgeblendet. Sobald ein *scratch*-Programm gespeichert wird, wird jedem offenen Skript der aktuelle Nutzer als Autor zugeordnet. Solche bzw. ähnliche Darstellungsformen der Historie von Autoren finden sich häufig in Entwicklungsumgebungen, z.B. auch in *Squeak* selbst.

Die Autorenliste, die im Kontextmenü der Skripte dargestellt wird, setzt voraus, dass der jeweilige Nutzer sich zu Beginn der Einheit identifiziert. Daher existiert im virtuellen Sandkasten ein Login Dialogfenster, welches vom Nutzer einen Nutzernamen und ein Passwort abfragt (siehe Abbildung 8.5). Der Nutzername wird als Autorenname bei jeder Aktion in einer Liste gespeichert, die dann im jeweiligen Kontextmenü zu betrachten ist. Das Passwort wird derzeit nicht geprüft, es stellt eine Möglichkeit für einen späteren Zeitpunkt dar, um z.B. zu einer geschützten Webplattform gleich als angemeldeter Nutzer wechseln zu können. In der aktuellen Version wäre die Übertragung jedoch unverschlüsselt, so dass davon abgesehen wurde, dies umzusetzen.

[6] http://www.mediawiki.org/wiki/MediaWiki, zuletzt besucht am 15. April 2012.

[7] *Mediawiki* ist beispielsweise die von Wikipedia verwendete *Wikiengine* und somit weit verbreitet.


Abbildung 8.5: Der Login-Dialog des virtuellen Sandkastens, um jeweiligen Nutzernamen und zugehöriges Passwort zu Beginn einzutragen.

### 8.1.5 Implementation

#### Squeak Grundgerüst

Die Klasse `ScratchFrameMorph` stellt einen Ausgangspunkt für die Implementation dar. Ein Exemplar dieser Klasse hält sämtliche grafischen und funktionellen Elemente zusammen. Sie bettet die Bausteinbibliothek (`ScratchLibraryMorph`), den Skripteditor (`ScratchScriptEditorMorph`) und die Bühne (`ScratchViewerMorph`) ein und erlaubt den Zugriff auf deren Methoden. Sämtliche Funktionen von grafisch repräsentierten Knöpfen sind ebenfalls im `ScratchFrameMorph` als Methodenaufrufe hinterlegt. Die Skript-Bausteine werden in unterschiedlicher Ausprägung (Vererbung) von der Klasse `BlockMorph` definiert.

#### Die Anpassungen

Zunächst wurden die drei Knöpfe in die Oberfläche von *scratch* eingebunden und der Optik bzw. Ästhetik angeglichen. Je Knopf wurden fünf unterschiedliche Grafiken für die verschiedenen Zustände im GIF[8]-Format erzeugt (siehe Abbildung 8.2) und in die *Squeak*-Umgebung eingeladen. Daraufhin wurden im Quelltext von *scratch* einige Änderungen vorgenommen: In der zum Programmstart aufgerufenen Methode `createButtonPane` der Klasse `ScratchFrameMorph` werden die drei neuen Knöpfe visuell eingebettet und mit den entsprechenden Methoden verknüpft. Die Methoden `sandboxToolTig`, `twitterToolTig` und `wikiToolTiG` zu der Klasse `ScratchFrame-`

[8]GIF steht für das *Graphics Interchange Format*.

`Morph` stellen die jeweiligen Funktionalität der Knöpfe zur Verfügung und werden ausgeführt, sobald der jeweilige Knopf betätigt wurde.

### Funktionalität Gruppenplattform

Die Methode `sandboxToolTig` erzeugt bei erfolgreicher Überprüfung[9], ob RSS-Feeds bzw. eine Internetverbindung zur Verfügung stehen, ein neues Objekt der Klasse `SandboxDialog`, welche wiederum beim Initialisieren die eigene Methode `showNewsFeed` aufruft, um ein Dialogfenster mit Knöpfen zum Schließen und zum Besuchen des CommSy-Raums zu erstellen. In einem `ScrollingStringMorph` des Dialogfensters werden sämtliche RSS-Benachrichtigungen des entsprechenden CommSy-Raums dargestellt. Die klasseneigene Methode `getUserResponse` verarbeitet die Eingaben des Nutzers.

Der im Hintergrund laufende Dienst `RSSStepper` wurde für den virtuellen Sandkasten entwickelt, um regelmäßig RSS-Streams auf Neuigkeiten überprüfen zu können und diese dann als vollständige Textdatei im Dateisystem zu speichern. Beim Start des virtuellen Sandkastens wird geprüft, ob seit der letzten Programmausführung Neuigkeiten aufgetaucht sind. Ist dies der Fall, wird die Variable `sandboxNews` auf `true` gesetzt, so dass dies von der im Hintergrund von `ScratchFrameMorph` regelmäßig aufgerufenen Methode `updateSandboxButtons` erkannt wird, um den Zustand des Knopfes anzupassen. Eine nötige Änderung des Zustands der Variable `sandboxNews` wird in gleicher Weise während der Laufzeit des Programms alle drei Minuten durchgeführt und durch den `RSSStepper` in Kombination mit einem *Python*-Skript (siehe S. 142) sichergestellt. Die gespeicherten Textdateien werden jeweils in das Dialogfenster eingebunden.

### Funktionalität Gruppenchat

Um von einer Applikation über die angebotene *Twitter* API auf *Twitter*-Dienste bzw. -Konten zugreifen zu können, muss eine Authentifizierung mittels *OAuth*[10] implementiert werden. Dieser Standard stellt sicher, dass ein Nutzer einer Applikation den Zugriff auf sein privates Online-Konto (hier *Twitter*) erlaubt. Dabei wird gleichzeitig gewährleistet, dass sensible Nutzerdaten nicht über die Applikation, sondern direkt an die jeweilige Onlineanwendung gesendet werden und so nicht an Dritte geraten können. Dies macht es jedoch nötig, dass der Nutzer eine Bestätigungspin

[9] Bei nicht vorhandener Verbindung zum RSS-Server wird die Aktion mit einem Hinweis auf die fehlende Verbindung abgebrochen.

[10] http://oauth.net/, zuletzt besucht am 15. April 2012.

von *Twitter* über einen Webbrowser erhält, den er in der Applikation angeben muss, um dieser die Anbindung an *Twitter* zu erlauben. Ist dieser Zugang für die Applikation hergestellt, können darin sämtliche Funktionen und Aktionen von *Twitter* durch Implementation der entsprechenden Methoden der *Twitter* API bereitgestellt werden. Die Anbindung an die API im Allgemeinen und die Implementation von *OAuth* im Speziellen war nicht in vertretbarer Zeit als native Lösung in *Squeak* zu realisieren. Daher wurde eine Anbindung von *Squeak* an ein im Hintergrund laufendes *Python*-Skript entwickelt, welches wiederum eine gute Anbindung an die API und *OAuth* über entsprechende *Python*-Bibliotheken lieferte (siehe unten, Abschnitt 8.1.6). Da sich diese Webschnittstelle als praktikabel und stabil erwies, wurde auch die gesamte Prüfung auf Neuigkeiten bei RSS-Feeds in *Python*-Skripte ausgelagert.

Zur Kommunikation zwischen *Squeak* und *Python* wird eine Socket-Verbindung hergestellt. Dazu wird zunächst ein *scratch*-Server initialisiert, der mit einer eindeutigen IP-Adresse auf einem bestimmten Port Nachrichten verschicken und empfangen kann (Host: 127.0.0.1, Port: 42001). Das im Hintergrund gestartete *Python*-Skript „ScratchSandboxClientTweepy“ kann ebenfalls über diese Verbindung Nachrichten empfangen und verschicken. Diese Server-Funktionalität existiert bereits rudimentär im *scratch* Quelltext, so dass an der Nachrichtenstruktur Änderungen vorgenommen werden mussten, die die neuen Funktionalitäten wie *Twitter* Authentifizierung und RSS-Abfrage ermöglichen. Darüber hinaus wurde die Methode `sandboxOnlineCheckMessage` hinzugefügt, die in regelmäßigen Zeitabständen durch den `RSSStepper` eine Nachricht sendet, um eine erneute Prüfung auf Neuigkeiten bei den RSS-Feeds über *Python* zu veranlassen. Die Methode `processIncoming-SandboxNews` wurde implementiert, um auf die eingehenden Mitteilungen des *Python*-Skripts eingehen zu können und die jeweiligen Variablen des virtuellen Sandkastens entsprechend zu setzen und auf Neuigkeiten in dem jeweiligen RSS-Feed zu reagieren (Darstellung über die Knöpfe).

Die Authentifizierung über *OAuth* wurde wie folgt umgesetzt: Nach einer Anmeldung der Applikation bei *Twitter* erhält man zwei eindeutige Tokens, die man zur Verifizierung der Applikation benötigt. Diese erlauben es wiederum, eine Webseite von *Twitter* anzusteuern, auf der ein Nutzer gebeten wird, der Applikation den Zugriff auf sein Konto zu erlauben. Stimmt er diesem durch Angabe seines Nutzernamens und des Passwort zu, generiert die Webseite eine PIN, die der Nutzer dann in der Applikation eingeben muss und die dann wiederum mit *Twitter* synchronisiert wird. Beim aktuellen System wird im Hintergrund zu Beginn der Ausführung die Verbindung zu *Twitter* im *Python*-Skript hergestellt, ein Webbrowser mit entspre-


Abbildung 8.6: Das Dialogfenster zur PIN-Eingabe (Authentifizierung *Twitter*).

chender Webseite geöffnet und eine Benachrichtigung an den virtuellen Sandkasten verschickt. Auf diese Nachricht wird innerhalb des virtuellen Sandkastens mit der Methode `getTwitterPin` reagiert. Diese initialisiert ein Dialogfenster zur Eingabe der PIN, den der Nutzer im Webbrowser erhält (siehe Abbildung 8.6). Sobald diese Eingabe durch den Nutzer getätigt wurde, wird eine Nachricht an das *Python*-Skript mit der PIN versandt, der diesen wiederum im Hintergrund an *Twitter* weiterleitet und somit den Authentifizierungsvorgang abschließt. Sämtliche Zugriffe auf *Twitter* sind nun für diese Instanz und für den Nutzer zulässig. Dies heißt, dass in dem Dialogfenster des Gruppenchats Nachrichten analog zur Implementation des Dialogfensters der Gruppenplattform gelesen werden können. In dem Dialogfenster, also dem Exemplar von `TwitterDialog`, wird darüber hinaus ein Exemplar der neu hinzugefügten Klasse `NoScrollingStringMorph` eingebettet, das es erlaubt, in einem Textfeld eine Nachricht zu verfassen mit maximal 140 Zeichen[11], die dann über den entsprechenden Knopf des Dialogfensters über den *Twitter* Dienst verbreitet wird. Hierzu wird die Nachricht im Hintergrund zur Anbindung an *Twitter* an das *Python*-Skript weitergeleitet. Die durch den entsprechenden Knopf aufgerufene Methode `sendTweet` prüft jedoch vorher, ob die Nachricht womöglich schon einmal versandt wurde. In jedem Fall wird der Nutzer informiert, ob die Nachricht verschickt wurde. Darüber hinaus enthält die Methode eine Verzögerung von 10 Sekunden, bis das Textfenster wieder zur Bearbeitung, also dem Verfassen einer weiteren Nachricht, frei gegeben wird.

[11] *Twitter* Nachrichten sind begrenzt auf 140 Zeichen.

### Funktionalität Wiki

Da die Wikibenachrichtigung bzw. die Funktionalität des Knopfs einen Kontextwechsel beabsichtigte, war hier ein einfaches Vorgehen angebracht: Analog zu den oben beschrieben Verfahren wird über das *Python*-Skript das angebundene Wiki auf aktuelle Aktivitäten geprüft. Sind Neuigkeiten vorhanden, ändert sich wie oben beschrieben das Aussehen des entsprechenden Knopfs. Dieser erlaubt durch die Verknüpfung mit der Klassenmethode `primOpenURL: aString` den direkten Sprung zur entsprechenden Wikieinstiegsseite im jeweiligen Standardwebbrowser.

### Kontextmenü


Abbildung 8.7: Das Kontextmenü eines Skripts. Die Nutzerin „Lea“ war die letzte Autorin dieses Skripts. „Überblick Autoren“ öffnet ein Dialogfenster, welches die gesamte Autorenhistorie anzeigt.

Die Funktionalität der Skript-Kontextmenüs wurde in der Klasse `HatBlockMorph` angepasst: Hier existieren nun Methoden, die eine chronologische Liste mit Autorennamen speichern (Duplikate werden zugelassen, sofern nicht zwei aufeinander folgen). Die Methoden `lastAuthor` und `showHistory` liefern den letzten Autoren und die Liste aller Autoren. Der letzte Autor erscheint immer als aktueller Besitzer im Kontextmenü, das über die Methode `rightButtonMenu` der Klasse `BlockMorph` erzeugt wird (siehe Abbildung 8.7). Unter dem letzten aktuellen Autoren gibt es im Kontextmenü die Möglichkeit, sich die Historie aller Autoren des Skripts in einem neuen Dialogfenster anzeigen zu lassen (siehe Abbildung 8.8). Die dortige Darstellung der Autoren berücksichtigt auch die Bearbeitungszeitpunkte. In mehrstufigen Angaben wird so dem Nutzer deutlich gemacht, wann Änderungen von wem durch-

geführt wurden. Die Bezeichnungen für die Zeitabstände der Änderungen werden durch die Methode `asStringWithCrAndTimeValue: anHatBlockMorph` dynamisch an die aktuelle Uhrzeit und das Datum angepasst; sie lauten: „Vor wenigen Minuten“ (weniger als fünf Minuten), „In der letzten Stunde“, „Vor wenigen Stunden“ (weniger als 5 Stunden), „Am heutigen Tag“, „Vor wenigen Tagen“ (weniger als drei Tage), „Innerhalb von 10 Tagen“, „Im vergangenen Monat“, „In den letzten 5 Monaten“ und „Älter als 5 Monate“.

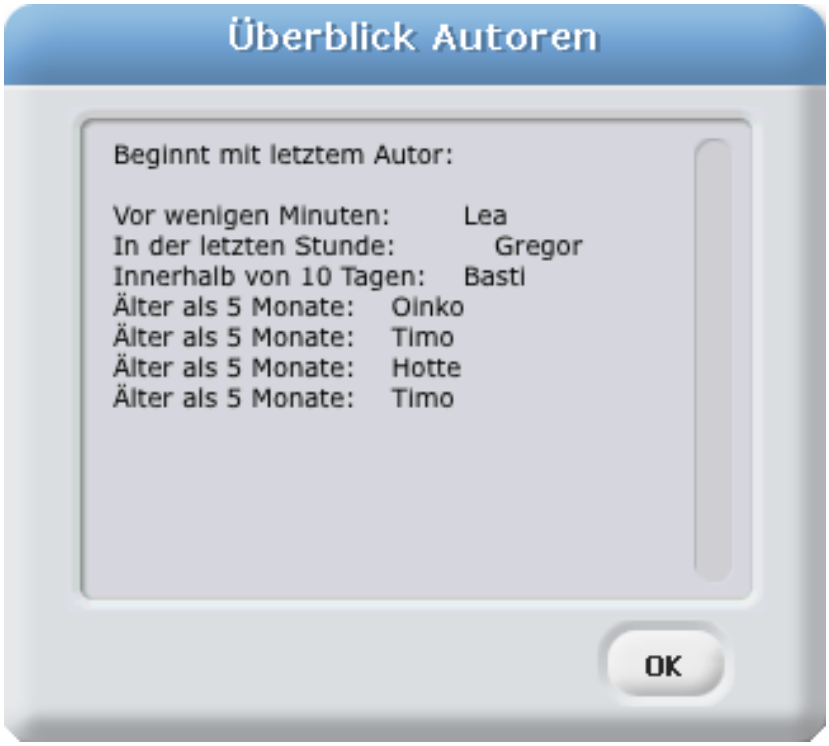



Abbildung 8.8: Das Dialogfenster zeigt alle Autoren des ausgewählten Skripts. Die Liste ist chronologisch geordnet und gibt ungefähre Zeitabstände der Änderungen zu aktueller Uhrzeit und Datum wieder.

### Login Dialog

Bei der Initialisierung des virtuellen Sandkastens wird ein Exemplar des modalen Dialogfensters `SandboxLoginDialog` über die `startup`-Methode des `ScratchFrameMorph` erzeugt. Das Dialogfenster erwartet eine Eingabe der Nutzerdaten im jeweiligen `StringFieldMorph` (siehe nochmals Abbildung 8.5). Die eingegebenen Daten werden beim Betätigen des OK-Knopfs ungeprüft an den entsprechenden Stellen im globalen `loginDict` hinterlegt.

#### Serialisierung

Sowohl die Autorenübersicht der Skripte im Kontextmenü als auch die Überprüfung auf Neuigkeiten bei RSS-Feeds müssen Funktionen sein, deren Inhalte beim Abspeichern, Beenden und Öffnen von *scratch*-Programmen nicht verloren gehen. Daher mussten Änderungen an der Serialisierung von *scratch* vorgenommen werden. Dazu wurde in der Klasse `HatBlockMorph` die Methode `checkForChangeAndSave: aNewAuthor` hinzugefügt. Diese wird beim Speichern eines Programms vom `ScratchFrameMorph` über `writeScratchProject` aufgerufen. Die Methode `checkForChangeAndSave: aNewAuthor` prüft für jeden vorhandenen `HatBlockMorph`, ob er ein Skript darstellt (ein oder mehrere verknüpfte Bausteine) und ob bereits Skriptautoren existieren. Tritt mindestens ein Fall ein, so wird der neue Autor abgespeichert, sofern er nicht bereits der letzte aktuelle Autor ist.

### 8.1.6 Ausführbarkeit des Prototyps

Der Arbeit liegt der gesamte Quelltext bei („ScratchSource"-Ordner inklusive *Python*-Unterordner). Der Prototyp wurde auf einem MacBook Pro 2,16 GHz Intel Core 2 Duo unter *Mac Os X* 10.5.8 mit *Python* 2.5.1, einem MacBook Air 2.13 GHz Intel Core 2 Duo, sowie einem Mac mini 2 GHz Intel Core 2 Duo unter *Mac Os X* 10.6.8 mit *Python* 2.6.1 getestet. Es werden folgende zusätzlichen Bibliotheken für *Python* benötigt: *Tweepy*[12], *oauth*[13], *Simplejson*[14] und *Universal Feed Parser*[15]. Für Mac-Umgebungen existiert im „ScratchSource"-Ordner eine ausführbare Anwendungsdatei namens „StartVirtualSandbox", die sämtliche benötigte Prozesse bzw. Programme startet; in diesem Startskript muss jedoch vor Benutzung der Pfad bzw. das Heimverzeichnis manuell angepasst werden. Grundsätzlich sind alle verwendeten Programmierumgebungen plattformunabhängig und sollten sich dementsprechend zumindest theoretisch auf *Microsoft Windows* Systemen starten lassen (*Unix/Linux* benötigt höchst wahrscheinlich grundlegende manuelle Änderungen, da sich der Quelltext von *scratch* für *Linux* von der Mac OS/Windows-Version unterscheidet und zum Zeitpunkt des Schreibens dieser Arbeit nicht offiziell quelloffen zur Verfügung steht). Aufgrund der Anpassungen bei der Serialisierung können abgespeicherte Programme des virtuellen Sandkastens auch nur darin wieder geöffnet

[12] https://github.com/tweepy/tweepy, zuletzt besucht am 15. April 2012.
[13] http://code.google.com/p/oauth, zuletzt besucht am 15. April 2012.
[14] http://undefined.org/python/#simplejson, zuletzt besucht am 15. April 2012.
[15] http://code.google.com/p/feedparser/, zuletzt besucht am 15. April 2012.

werden. Versuche, dies mit der offiziellen *scratch*-Version durchzuführen, können zu Programmabstürzen führen.

In der beigelegten Version startet der virtuelle Sandkasten im *user mode*, der unveränderlich das Fenster der *scratch*-Oberfläche präsentiert und so eine Interaktion mit der zugrunde liegenden *Squeak*-Entwicklungsumgebung verhindert. Ein Mausklick bei gedrückter Shift-Taste auf „Extras“ erlaubt das Verlassen des *user modes* und gibt damit die Entwicklungsumgebung und die Klassenstruktur des Quelltexts im *Squeak*-Browser frei. Darin können alle vorgestellten Implementationsschritte nachvollzogen werden. Momentan sind sämtliche Verbindungen zu den einzelnen Webschnittstellen statisch, diese können bzw. müssen manuell auf den jeweiligen Kontext angepasst werden. Dies geschieht über die Initialisierung der Variablen `commsy` und `wiki` mittels der Methodenaufrufe `FeedFramework.feed(url)` im *Python*-Skript „ScratchSandboxClientTweepy“. Um einen Tweet festzulegen, muss für diesen über *OAuth* ein zugehöriger Schlüssel erstellt werden, um diese Werte dann im Methodenaufruf `OAuthHandler(token, secret)` des Skripts „TweepyFramework“ als Parameter zu übergeben[16].

Der Prototyp kann in englischer und deutscher Lokalisation verwendet werden, die Lokalisationsdatei für die deutsche Sprache ist im Unterordner „locale“ zu finden („de.po“). Die Anpassung der Sprache kann direkt in der Menüleiste des virtuellen Sandkasten vorgenommen werden.

## 8.2 Agile Methoden im Schulkontext

Die Funktionen des virtuellen Sandkastens betten Gruppenaktivitäten in die *scratch*-Entwicklungsumgebung ein. Diese Prozesse werden entsprechend des *blended learning* mit angebrachtem methodisch-didaktischen Vorgehen aufgegriffen und strukturiert, um eine angebrachte Lernumgebung zu gewährleisten. Eine Anlehnung an die moderne Softwareentwicklung ist empfehlenswert, da so Lehrern und Betreuern starke Argumente geliefert werden, um die Akzeptanz unter Schülern z.B. für die Verwendung von Computern in Paaren zu erhöhen (vergleiche Kapitel 3.1.1).

Darüber hinaus sollen die Anforderungen aus Kapitel 7.1 ebenfalls von den agilen Methoden abgedeckt werden. Dabei unterstützen die agilen Methoden verstärkt aber nicht ausschließlich die Aufbauphasen von Gruppen (vergleiche ebenfalls Johansen et al. [1991]; Janneck und Janneck [2004] in Kapitel 3.2.1).

[16] Es ist anzunehmen, dass *Twitter* zukünftig *OAuth* 2.0 verlangen wird und die hier vorgestellte Authentifizierung nicht mehr unterstützt wird bzw. angepasst werden muss.

Der folgende Text bis Abschnitt 8.2.4 wurde Göttel [2011a] entnommen und geringfügig angepasst bzw. erweitert. Es handelt sich dabei um nötige didaktische, methodische und strukturelle Anpassungen für den Einsatz im Schulkontext von *pair programming*, *user stories*, *informative workspace*, *meetings* und *collective code ownership*.

Programmiertechnische Vorgehensweisen der agilen Methoden wurden in Schulkontexten bereits erfolgreich eingesetzt [Weigend, 2005] (siehe Kapitel 3.2). Im Gegensatz dazu werden hier nun die sozialen Aspekte der Arbeitsprozesse der agilen Methoden näher betrachtet, die bereits in Kapitel 2.2 vorgestellt wurden. Die vorgestellten Methoden aus XP und Scrum wurden in den Projekten aus Kapitel 6 an Schulen und ähnlichen Einrichtungen erfolgreich verwendet. Die Methoden werden nachfolgend mit Hinweisen versehen, wie die jeweilige Methode in einem Schulkontext anzuwenden ist, welche Probleme auftreten können und wie diesen entgegengewirkt werden kann.

### 8.2.1 Pair programming

Im Schulkontext ist Wert darauf zu legen, dass die Rolle der Person ohne Tastatur ernsthaft eingenommen wird und es auch wirklich zum Wechsel der Tastatur kommt. Hierfür ist eine zeitliche Taktung hilfreich. Darüber hinaus ist bei Paaren mit stark unterschiedlichem Wissenstand darauf zu achten, dass die Person mit einem geringeren Kenntnisstand die Tastatur häufiger verwendet und der Partner sich darin übt, sein Wissen zu vermitteln. Hierbei ist besonders zu berücksichtigen, dass nicht einfach die Anweisungen diktiert werden, sondern Konzepte erklärt werden. Notfalls kann es helfen, Paare zu tauschen (vergleiche McKinney und Denton [2005], Kapitel 3.1.1).

In mehreren Projekten konnte beobachtet werden, dass gerade Schülerinnen das Programmieren im Paar schnell annehmen und häufig davon begeistert sind, dass gemeinsam mit einem Partner Lösungen erarbeitet werden. Dies ist einer von vielen Schritten, um weibliche Jugendliche für IT-Berufe zu begeistern. Studien mit weiblichen Studierenden lassen zumindest die Hoffnung zu, dass solche sozialen Aspekte der Softwareentwicklung das Selbstbewusstsein bezüglich der eigenen Informatikfähigkeiten stärken, welches benötigt wird, um sich das Ergreifen eines IT-Berufs zuzutrauen [Berenson et al., 2004].

### 8.2.2 User stories

Im Schulkontext handelt es sich bei *user stories* um eine Methode, die gerade zu Anfang eine Hilfestellung benötigt, da Schüler häufig dazu tendieren, möglichst alle Funktionalitäten in einer Beschreibung abdecken zu wollen. Elementare Aufgaben erscheinen ihnen schnell als zu banal. Daher empfiehlt es sich für den Anfang, Beispiele vorzugeben und gemeinsam mit den Teilnehmern zu erarbeiten. Diese kleinteilige Herangehensweise erleichtert die Gruppenarbeit, da nur so Aufgaben verteilt werden können und niemand befürchten muss, dass immer nur andere Teilnehmer die spannenden und fordernden Aufgaben vermittelt bekommen, wohingegen sie selbst entweder zu banale oder aber unlösbare Aufgaben zugeteilt bekommen.

### 8.2.3 Informative workspace


Abbildung 8.9: Die Poster des *informative workspace* können als Diskussionsgrundlage zur Präsentation und zur Dokumentation verwendet werden. Die Namen der Teilnehmer wurden nachträglich anonymisiert.

Den meisten Schülern ist auf Anhieb nicht ersichtlich, wozu sie solche Poster anfertigen sollen. Es empfiehlt sich daher, ihnen klar zu machen, dass Poster die Arbeitswege darstellen und somit gut verwendet werden können für mögliche Präsentationen und Abschlussdokumentationen (siehe beispielsweise Abb. 8.9). Es ist

aber auch zu erwähnen, dass die Poster helfen können einen Überblick zu behalten, Ziele zu vergegenwärtigen und den Fokus beizubehalten. Es ist empfehlenswert für die Veranstaltungsleiter, von Zeit zu Zeit auf die Poster einzugehen, gezielt Fragen zu stellen, ob diese noch aktuell sind, und Diskussionen anhand der Poster zu führen.

### 8.2.4 Meetings

Hier muss gerade zu Beginn stark darauf geachtet werden, dass jede Person individuelle Erfahrungen mitteilt. In der Klassensituation entsteht häufig eine Dynamik, in der auf den Vorredner verwiesen wird. Dies ist möglichst zu unterbinden, indem man deutlich macht, dass in der Gruppe jeder für das gemeinsame Ziel arbeitet und der Erfolg davon abhängt.

### 8.2.5 Collective code ownership

Diese Praktik kann dazu verleiten, dazugehörige Ideologien und Pflichten konzeptuell vermitteln zu wollen bzw. diese so stark in den Vordergrund zu stellen, dass sie hemmend auf Schüler wirken. Die Gedanken des *collective code ownership* sind zwar wichtig in der professionellen Softwareentwicklung zu befolgen, jedoch sind sie für den Informatikunterricht an Schulen deutlich überfrachtet. Es ist daher darauf hinzuweisen, dass alle Teilnehmer Änderungen vornehmen können, diese jedoch eindeutig einem Autoren zugeschrieben werden, um bei auftretenden Problemen sämtliche Autoren eines Skripts nach durchgeführten Änderungen befragen zu können. Auf diese Weise kann sichergestellt werden, dass sich die gesamte Gruppe für die Skripte zuständig fühlt. Demnach kann vielleicht von einer *collective code responsibility* gesprochen werden.

### 8.2.6 Anwendbarkeit der angepassten agilen Methoden

Die häufig leicht und schnell umzusetzenden Anpassungen bei den genannten Praktiken der agilen Methoden machen deutlich, dass sie sich gut für den Einsatz im Schulkontext eignen. Dabei ist es hilfreich, auf den Praxisbezug der Praktiken hinzuweisen und die dadurch entstehenden sozialen Interaktionen der Teilnehmer zu loben. Das Nennen von Beispielen bekannter größerer Softwareentwicklungshäuser, die agile Methoden einsetzen, hilft bei der Einführung der Praktiken der agilen Methoden bzw. kann die Motivation steigern.

## 8.3 Kapitelzusammenfassung

Dieses Kapitel präsentierte den virtuellen Sandkasten und die angepassten agilen Methoden. Der virtuelle Sandkasten erweitert *scratch* um Anbindungen zu einer Gruppenplattform (CommSy), einem Gruppenchat (*Twitter*) und einem Wiki (Mediawiki). Zusätzlich steht eine neue Funktionalität des Kontextmenüs zu den Skripten zur Verfügung, die erlaubt, dass eine Autorenhistorie zu dem jeweiligen Skript betrachtet werden kann. Sämtliche Änderungen wurden mit *Squeak* und *Python* realisiert.

Bei den Praktiken der agilen Methoden wurden Anpassungen an *pair programming*, *user stories*, *informative workspace*, *meetings* und *collective code ownership* vorgestellt. Die Anpassungen sind zumeist methodischer und struktureller Art und benötigen nur wenig Aufwand seitens des Betreuers und der Teilnehmer, um sie erfolgreich im Schulkontext anzuwenden. Ein erfolgreicher Einsatz der agilen Methoden im Schulkontext kann auf diesem Wege sichergestellt werden.

Der virtuelle Sandkasten und die angepassten agilen Methoden sind als gegenseitige Ergänzung zu verstehen. Im folgenden Kapitel werden die Anforderungen aus Kapitel 7.1 bezüglich der Abdeckung durch Kombination aus virtuellem Sandkasten und agilen Methoden geprüft. Darüber hinaus werden aus Sicht jugendlicher Personas die erfahrbaren sozialen Aspekte beim virtuellen Sandkasten und den agilen Methoden bewertet.

# Kapitel 9

# Abdeckung der sozialen Aspekte

Zum jetzigen Zeitpunkt wurde in der vorliegenden Arbeit offengelegt, dass sich Jugendliche zu einfach ein einseitiges Bild der Informatik sowohl im gesellschaftlichen als auch im Schulkontext aneignen. Die vorgestellten Projekte aus Kapitel 6 haben gezeigt, dass es mit aktuellen (und erfolgreichen) ILEs nach wie vor schwer ist, auf soziale Dynamiken in der Softwareentwicklung hinzuweisen, da es an computergestützten Lösungen fehlt. Die Betrachtungen zu Unterrichtsformen und die Lehrerbefragung in Kapitel 3.2 ergaben, dass es ebenfalls an didaktischen Methoden fehlt, um Programmierprojekte mit Schülern so zu gestalten, dass sie soziale Abläufe der professionellen Softwareentwicklung explizit widerspiegeln. Dies würde Teilnehmer darin unterstützen, sich ein wirklichkeitsnahes Bild der Informatik zu machen. Sowohl der virtuelle Sandkasten als auch die angepassten agilen Methoden sollen dieses Unterfangen unterstützen.

Das folgende Kapitel untersucht die konkrete Abdeckung der erarbeiteten Lösungen zu einer umfassenderen, also sozialen und dynamischen, Sichtweise auf die Softwareentwicklung. Dazu werden zunächst die erarbeiteten Konzepte und Systeme mit den in Kapitel 7.1 aufgestellten Anforderungen zur Stärkung der sozialen Aspekte abgeglichen. Um eine weitere Betrachtung aus Sicht der Schüler anstellen zu können, werden darüber hinaus jugendliche Personas vorgestellt, die aus den Projektbeobachtungen entstanden sind. Mit diesen ist es möglich, aus unterschiedlichen Blickwinkeln die Attraktivität von ILEs oder auch Unterrichtsmethoden zu bewerten. Sie werden eingesetzt, um jeweils *cognitive walkthroughs* für *scratch* im klassischen Gruppenunterricht und im virtuellen Sandkasten mit den angepassten agilen Methoden durchzuführen. Darüber hinaus werden die jugendlichen Personas

eingesetzt, um die beiden Varianten in einer Gewichtungsmatrix bewerten zu können.

## 9.1 Überprüfung der Anforderungen zur Stärkung der sozialen Aspekte

Die ermittelten Anforderungen zur Stärkung der sozialen Aspekte aus Kapitel 7 werden hier nun für den virtuellen Sandkasten und die angepassten agilen Methoden herangezogen und auf ihre theoretische Abdeckung überprüft. Zu den Anforderungen wird jeweils erläutert, auf welche Weise diese mittels der technischen Erweiterungen und der methodisch-didaktischen Vorgehensweisen erreicht werden und ob weitere Potenziale zur Fokussierung auf soziale Aspekte durch technische oder methodische Unterstützung vorhanden sind. Für einen schnellen Überblick, welche einzelnen Aspekte die Anforderungen erfüllen, sorgt Tabelle 9.1.

| | Virtueller Sandkasten | PP | US | IW | M | CCO |
|---|---|---|---|---|---|---|
| Bezugspunkt Gruppe | Gruppenplattform, Gruppenchat | | | ✓ | ✓ | |
| Kollektive Ressourcen | Gruppenplattform | | | ✓ | | |
| Gemeinschaftsgefühl | Gruppenplattform, Gruppenchat, Wiki, Kontextmenü | ✓ | ✓ | ✓ | ✓ | ✓ |
| Wissensaustausch | Gruppenplattform, Gruppenchat | ✓ | | ✓ | ✓ | |
| Kenntnis Projektfortschritt | | | | ✓ | ✓ | |
| Internes Tutorennetzwerk | Gruppenplattform, Gruppenchat | | | ✓ | | |
| Programmvisualisierung | (Gruppenplattform, Wiki) | | | ✓ | | |
| Zusätzliche Anreize | Wiki | | | | | |
| Vielfältigkeit | Gruppenplattform | | ✓ | ✓ | | |

Tabelle 9.1: Erfüllte Anforderungen zur Stärkung der sozialen Aspekte des virtuellen Sandkastens und der agilen Methoden (PP = *pair programming*, US = *user stroies*, IW = *informative workspace*, M = *meetings*, CCO = *collective code ownership*).

Allgemein kann festgestellt werden, dass die sozialen Aspekte nach Fisker et al. [2008] im virtuellen Sandkasten mit den angepassten agilen Methoden tatsächlich,

wie in Kapitel 7.2 gefordert, berücksichtigt werden: Einen Austausch von Artefakten stellt die Onlineplattform sicher, zu der eine Schnittstelle im virtuellen Sandkasten existiert. Darin können ebenfalls Diskussionen und Anmerkungen hinzugefügt werden, die der Kommunikation förderlich sind. Des Weiteren sorgt auch der Gruppenchat dafür, dass Kommunikation gefördert wird. Durch die Echtzeitfunktionlität (bzw. nur leicht verzögerte Kommunikationsform) des Gruppenchats wird *awareness* für die Aktivitäten innerhalb der Gruppe sichergestellt. Prinzipiell ist es vorstellbar, dass alle vier Kategorien der *awareness* (informelle *awareness*, soziale *awareness*, *awareness* über Gruppe/Struktur und *awareness* über Arbeitsplatz; vergleiche Kapitel 3.1) durch den Gruppenchat abgedeckt werden. Dies hängt jedoch stark von dem wirklichen Nutzerverhalten ab. Somit können in einem Schulkontext ohne konkrete Anleitung nicht alle Kategorien abgedeckt würden. Eine nahe liegende Behauptung wäre, dass besonders die informelle *awareness* durch den Gruppenchat transportiert wird, da es beispielsweise für Schüler selbstverständlich ist, in einem Chat die eigenen aktuellen Aktivitäten mitzuteilen. Solch ein aktives Verhalten wird durch die *meetings* und den *informative workspace* zusätzlich gestützt.

Ein Bezugspunkt für die Gruppe findet sich sowohl im virtuellen Sandkasten als auch im *informative workspace* und in den *meetings* wieder. Der virtuelle Sandkasten benachrichtigt über sämtliche Aktivitäten auf der Gruppenplattform (z.B. Termine, Diskussionen und Kommentare) und erlaubt einen schnellen Überblick sowie das direkte Aufsuchen der Plattform. Über den Gruppenchat können jederzeit innerhalb der Oberfläche im Gruppenkontext kurze Nachrichten versandt und empfangen werden. Der *informative workspace* ist jederzeit physisch zugänglich und dient als Diskussionsgrundlage, besonders in informellen Situationen. Die *meetings* geben jeden Morgen eine Einsicht in die Aktivitäten der einzelnen Teilnehmer. Auf diesem Wege wird jedem Teilnehmer während des gesamten Projekts ermöglicht, einen Bezugspunkt zur Gruppe herzustellen, sei es innerhalb der Oberfläche des virtuellen Sandkastens oder in zwischenmenschlichen Situationen. Hierdurch werden vielfältige gruppeninterne Austauschmöglichkeiten geboten, die ein Bewusstsein für die Gruppe entstehen lassen.

Kollektive Ressourcen werden besonders computerseitig abgedeckt, da es sich zumeist natürlich um digitale Artefakte handelt. Der virtuelle Sandkasten bietet die Möglichkeit, direkt die Gruppenplattform aufzusuchen, um dort Medien zu recherchieren oder diese dort abzulegen und so den anderen Teilnehmern zugänglich zu machen. Auch hierbei werden alle Teilnehmer über die Anpassung des Erscheinungsbilds des Knopfs informiert, sobald neue Medien auf der Plattform verfügbar sind.

Durch diverse Möglichkeiten, Diskussionen oder Kommentare zu den Medien zu hinterlassen, wird auch wieder das Bewusstsein für die Gruppe gestärkt, da dort alle Teilnehmer zu Wort kommen können. Der *informative workspace* bietet des Weiteren die Möglichkeit, Ideen, Planungen, Meinungen und Zwischenstände mitzuteilen und diese auch in computerfreien Situationen (z.B. beim Brainstorming) verfügbar zu machen.

Ein Gemeinschaftsgefühl soll möglichst mit allen agilen Methoden gefördert werden. Paare stehen in regem Austausch untereinander und kommunizieren mit anderen Paaren über den *informative workspace* und in den *meetings*. Auch die Erstellung von *user stories* ist ein gemeinschaftlicher Prozess, der viele Diskussionen erfordert. Die Funktionalität, Skripte anderer zu bearbeiten und jederzeit eine Historie der Autoren verfügbar zu haben, wird durch *collective code ownership* methodisch fundiert und durch das Kontextmenü jedes Skripts im virtuellen Sandkasten praktisch umgesetzt. Alle drei Funktionen der Knöpfe des virtuellen Sandkastens bieten den Gruppenmitgliedern Interaktion an und sind förderlich für das Gemeinschaftsgefühl.

Der Wissensaustausch erfolgt über die Gruppenplattform und den Gruppenchat des virtuellen Sandkastens in textlicher oder auch bildlicher Form. Dies kann sowohl zeitversetzt als auch ortsunabhängig stattfinden. Innerhalb der Paare findet verbaler Wissensaustausch vor Ort statt. Dieses Wissen wird wiederum schriftlich oder bildlich auf dem *informative workspace* repräsentiert und in *meetings* mündlich mitgeteilt. So werden zahlreiche Kanäle zum Wissensaustausch angeboten, die konservierende Eigenschaften besitzen, so dass zu späteren Zeitpunkten darauf zurückgegriffen werden kann.

Zwar ist prinzipiell ein Projektfortschritt auf der Gruppenplattform abzulesen, jedoch meist nur implizit. Entsprechend explizite Funktionen werden im aktuellen CommSy nicht von Schülern wahrgenommen bzw. durchdrungen. Daher beschränken sich Informationen zum Projektfortschritt auf den *informative workspace* und die *meetings*, die wiederum gut geeignet sind, um das Bewusstsein für die Gruppe im direkten Blickkontakt zu festigen.

Ein internes Tutorennetzwerk bildet sich Dank der Möglichkeit, auf der Gruppenplattform um Hilfe zu bitten oder diese dort anzubieten. Ebenfalls können im Gruppenchat in Echtzeit aktuelle Erfolge oder offene Fragen mitgeteilt werden. Während der *meetings* können auch Informationen zu gemeisterten Problemen und bei Bedarf konkrete Hilfe angeboten werden. Hierbei bilden sich Strukturen innerhalb der Gruppe, die zu einem bewussten Wahrnehmen des Gruppengefüges beitragen.

Neben der bereits vorhandenen einfachen Möglichkeit von *scratch*, Skripte eines Sprites als Bild zu exportieren, bietet der virtuelle Sandkasten momentan keine direkte Möglichkeit an, um ein Programm zu visualisieren. Ein möglicher indirekter Weg wäre, die exportierten Bilder mit den Skripten der Sprites weiter zu bearbeiten bzw. zusammenzuführen und diese dann wieder als Artefakt auf der Gruppenplattform oder dem Wiki zur Verfügung zu stellen. Von den agilen Methoden wird eine Programmvisualisierung bzw. Dokumentation durch den *informative workspace* sichergestellt. Die Plakate des *informative workspace* eignen sich auch als Präsentationsmittel.

Zusätzliche Anreize werden hauptsächlich über die Wiki-Funktionalität ermöglicht, da dort ein Austausch mit der Außenwelt angedacht ist; so können Schüler Verwandten und Bekannten ihre fertigen Ergebnisse präsentieren oder auch als Referenz verwenden. Idealerweise werden die Aktionen im Wiki bzw. die Außenpräsentation mit anderen Aufgaben oder gar Schulwettbewerben verknüpft. Allgemein sind diese Anreize also als Präsentationswerkzeug anzusehen.

Um Gruppenmitglieder ohne Programmierfähigkeiten oder -interessen einzubinden, werden neben den *scratch*-eigenen Anwendungen wie dem Malprogramm oder der Audioaufnahme Möglichkeiten zur Projektkoordination oder auch Dokumentation durch den *informative workspace* und die Gruppenplattform geboten. Darüber hinaus bieten *user stories* eine kreative Möglichkeit, schriftliche Dokumente zu verfassen, auf die die Gruppe zugreifen kann.

Diese Ausführungen machen deutlich, dass die gewählten Lösungen jeweils auf mehreren Ebenen die Anforderungen zur Stärkung der sozialen Aspekte aus Kapitel 7.1 erfüllen können und es so vermögen, die sozialen Aspekte des Entwicklungsprozesses adäquat und attraktiv darzustellen, um sie im weiteren Schulkontext wieder aufgreifen zu können. Darüber hinaus garantiert diese Vielfältigkeit an technischer Unterstützung und methodisch-didaktischer Praktiken formal *blended learning* (vergleiche Kapitel 4.2).

Auch die Eigenschaften, die laut Bruckman [1998] (Kapitel 4.2.2, S. 63) soziale Lernumgebungen anbieten müssen, werden durch den virtuellen Sandkasten und die angepassten agilen Methoden abgedeckt. So ist es jedem Nutzer möglich, positive Vorbilder zu identifizieren, mit diesen in Kontakt zu treten und gemeinsame Ziele anzugehen. Technikängste können durch ein gestärktes Gruppengefühl und Kommunikation zu auftretenden Problemen aller Teilnehmer gemindert werden. Der virtuelle Sandkasten und die angepassten agilen Methoden bieten zusätzlich die Möglichkeit für informellen und sozialen Austausch innerhalb der Gruppe.

Darüber hinaus ist nun ersichtlich, dass der virtuelle Sandkasten und die angepassten agilen Methoden in Kombination alle drei geforderten Phasen (vergleiche Kapitel 7), die in den Projekten beobachtet werden konnten, unterstützen. Sowohl die Aufbauphase, die Erhaltungsphase, als auch die Dokumentationsphase werden ausreichend und meist über verschiedene Kanäle unterstützt.

Es gilt jedoch, neben der rein analytischen Betrachtung noch weitere Einschätzungen aus der Schülerperspektive heranzuziehen. Analytische Evaluationsmethoden bieten den Vorteil, dass sie eine relativ ausgewogene, nachvollziehbare Argumentation aus unterschiedlichen Blickwinkeln erlauben. Eine überzogen affirmative Haltung und somit eine Verfälschung von Evaluationsergebnissen wäre nämlich zu erwarten, wenn Jugendliche in Interviews oder Fragebögen nach ihrer Meinung oder zur Attraktivität eines Systems befragt würden, das, ähnlich den Projekten aus Kapitel 6, von einem externen Gast vorgestellt würde (vergleiche dazu nochmals [Rosenthal und Jacobson, 1968; Rosenthal, 1995]). Daher werden im folgenden Abschnitt jugendliche Personas vorgestellt und zur analytischen Evaluation verwendet.

## 9.2 Personas als Gradmesser

Es ist anzunehmen, dass Nutzer häufig bereits durch die Oberflächengestaltung von Lernsystemen ungewollte und falsche Rückschlüsse auf die damit verbundene Disziplin ziehen [O'Malley, 1992, S. 345]. Folglich sind fehlende Hinweise auf soziale Aspekte in ILEs und didaktische Methoden besonders gravierend. Aus diesem Grund ist es ratsam, die entstandenen Lösungen der vorliegenden Arbeit einer weiteren Prüfung bezüglich der Oberfläche zu unterziehen. Dabei werden besonders die Äußerlichkeiten des virtuellen Sandkastens und der agilen Methoden aus Sicht der jugendlichen Nutzer betrachtet und bewertet. Hiermit wird ein Weg angeboten, der es erlaubt, aus unterschiedlichen Blickwinkeln zu argumentieren. In der Softwareentwicklung (genauer dem *Usability Engineering*) werden sogenannte Personas während des Entwicklungsprozesses als Argumentationsgrundlage verwendet, um Designentscheidungen aus verschiedenen Blickwinkeln entsprechend der definierten Zielgruppen zu treffen. In dieser Arbeit werden jugendliche Personas in einen neuen Kontext versetzt, der es so erlaubt, Oberflächen von Lernumgebungen oder didaktische Methoden auf ihre Außenwirkung hin zu untersuchen, um so Vermutungen über die wahrgenommene Attraktivität der IT anstellen zu können. Dieses in Kapitel 9.3 beschriebene Vorgehen wurde bei Göttel [2011b] vorgestellt und kann dort nachgelesen werden. Nachfolgend wird das herkömmliche Konzept der Personas in

der Softwareentwicklung allgemein und im Kontext CCI beschrieben. Darauf folgt eine Beschreibung vier jugendlicher Personas, die aus den in Kapitel 6.2 beschriebenen Projekte abgeleitet sind. In Abschnitt 9.3 werden diese dann herangezogen, um mittels *cognitive walkthroughs* und einer Gewichtungsmatrix *scratch* im klassischen Gruppenunterricht mit dem virtuellen Sandkasten und den angepassten agilen Methoden zu vergleichen.

### 9.2.1 Personas in der Softwareentwicklung

Personas wurden erstmals von Cooper [1999] für die Softwareentwicklung vorgeschlagen; sie stellen archetypische fiktive Nutzer dar, deren relativ ausgearbeitete Biografien und klare Wesenszüge schriftlich oder multimedial den Entwicklern zur Verfügung stehen. Diese Aufzeichnungen zu den Personas werden aus Beobachtungen und Datenmengen gewonnen. Gängige Merkmale einer Persona sind Name, Geschlecht, Alter, Porträt, Familiensituation, Freundeskreis, Ansichten, Vorlieben, Ängste, und Aversionen. Die Verwendung von Personas im gesamten Entwicklungsprozess ermöglicht es, Produkte zu schaffen, die möglichst vielen und verschiedenen Herangehensweisen, Bedürfnissen und Absichten der eigentlichen Nutzer entsprechen [Adlin und Pruitt, 2010]. Sie dienen dazu, innerhalb eines Entwicklungsteams explizit die Erfolge und Probleme der einzelnen Personas mit einem Systemprototyp anzusprechen und zu begründen [Pruitt und Grudin, 2003]. Personas sind im Gegensatz zu statistischen Daten durch ihre Namen und *Persönlichkeit* besser greifbar und daher als Argumentationsgrundlage in Diskussionen anschaulicher zu verwenden. Adlin und Pruitt fassen die Vorteile von Personas wie folgt zusammen: Personas schaffen eine gemeinsame Sprache, indem sie Annahmen und Wissen über Nutzer explizit veranschaulichen. Sie vereinfachen die Fokussierung auf spezielle Nutzer. Personas helfen Entwicklern, Empathie und Interesse für die Nutzer zu entwickeln.

Als zusätzliches Werkzeug im Zusammenhang mit Personas schlagen Adlin und Pruitt eine Gewichtungsmatrix vor, die die einzelnen Systemfunktionalitäten in Abhängigkeit von der Zielgruppenrelevanz der jeweiligen Personas repräsentieren soll [Adlin und Pruitt, 2010, S. 125 ff.]. Hierbei wird jeder Persona eine Gewichtung zugeteilt, um dann in der Matrix aus der Sicht jeder einzelnen Persona Notenwerte für die separierten Funktionalitäten zu vergeben. Die Notenwerte spiegeln den funktionellen und emotionalen Wert der jeweiligen Systemeigenschaften für eine Persona wider. Zur Vergabe der Noten sehen sie zwei gültige Wege. Aus externen Testnutzern oder auch internen Kollegen können Kleingruppen zusammengestellt werden, deren Teil-

nehmer den jeweiligen Personas in vielen Wesenszügen ähneln. Diese Kleingruppen vergeben dann nur Noten für die einzelnen Funktionalitäten aus Sicht der jeweiligen zugeordneten Persona. Eine andere Möglichkeit ist es, in einer Gruppe oder als Individuum aus der Sicht aller Personas die einzelnen Noten für die Eigenschaften zu vergeben. Als Notenbasis schlagen Adlin und Pruitt eine ganzzahlige Skala von -1 bis 2 vor:

2: Die Persona ist von der Eigenschaft begeistert, bzw. die Funktionalität unterstützt die Persona hervorragend (auch wenn sie es nicht realisiert).

1: Die Persona profitiert teilweise von der Eigenschaft (bemerkt/unbemerkt).

0: Der Persona ist die Funktionalität unwichtig.

-1: Die Funktionalität verwirrt, verärgert oder behindert die Persona.

Man kommt zu einer gewichteten Summe, indem die jeweiligen Notenwerte mit der Gewichtung multipliziert und über die Personas addiert werden (siehe Tabelle 9.2). Diese gewichtete Summe soll es Entwicklern ermöglichen, Prioritäten auf bestimmte Eigenschaften zu legen, um diese zu implementieren oder zu verfeinern. Es empfiehlt sich, in Abhängigkeit der Summen eine Rangfolge der wichtigsten Eigenschaften festzulegen. Die konkreten Werte der errechneten Summen haben jedoch keine weitere Aussagekraft, sie dienen nur zur Abgrenzung untereinander.

| | Persona A | Persona B | Persona C | Persona D | |
|---|---|---|---|---|---|
| Gewichtung | 40 | 40 | 15 | 05 | Gewichtete Summe |
| Eigenschaft 1 | 1 | 1 | 2 | 2 | 120 |
| Eigenschaft 2 | -1 | 0 | 1 | 0 | -25 |
| ... | | | | | |

Tabelle 9.2: Abstrakte Gewichtungsmatrix nach Adlin und Pruitt [2010].

Während das Hauptaugenmerk von Personas auf der Begleitung der Entwicklungsprozesse liegt, können sie auch zur Überprüfung eines angefertigten Prototyps oder eines fertigen Systems verwendet werden [Adlin und Pruitt, 2010, S. 151 ff.]. Hierzu schlagen Adlin und Pruitt den Einsatz von Personas in drei Bereichen vor. Zum einen sehen sie Personas geeignet für *cognitive walkthroughs* (siehe unten) und zur Überprüfung der gewählten Oberflächengestaltungen. Als zweite Möglichkeit nennen sie das Durchführen von Nutzertests aus Sicht der Personas. Als drittes

mögliches Einsatzgebiet definieren Adlin und Pruitt sogar die Qualitätssicherung, indem sie mit Hilfe der Personas Bugs auffinden wollen, da aus der Persona-Sicht häufig andere Wege als die eines Entwicklers durch ein System gewählt werden und so bisher nicht geprüfte Bereiche durchlaufen werden können.

Ein *cognitive walkthrough* ist eine analytische Evaluationsmethode, bei der Einzelpersonen oder auch Kleingruppen aus der Sicht der Nutzer wohldefinierte Aufgaben mit einer Oberfläche erledigen und diese auf Verständlichkeit und Erlernbarkeit prüfen [Wharton et al., 1994]. Personas helfen, die einzelnen Sichtweisen einzunehmen. Die durchführende Person konstruiert so für sämtliche Aufgaben und Personas Erfolgs- oder Misserfolgsgeschichten, um eine Aussage über das getestete System ableiten zu können.

### 9.2.2 Jugendliche Personas

Jugendliche Personas werden in der Literatur nur selten genannt. Antle [2006] schlägt diese als begleitendes Werkzeug während des Entwicklungsprozesses vor und nennt drei wesentliche Unterschiede zu erwachsenen Personas [Antle, 2006, 2008]:

- Jugendliche benutzen Anwendungen meist nicht, um einfach nur eine Aufgabe zu erledigen, sondern um Bedürfnisse zu befriedigen.
- Entwicklungsstufen und verbundene Fähigkeiten müssen stärker berücksichtigt werden als mögliche Arbeits- oder Aufgabenziele.
- Jugendliche legen einen deutlich größeren Wert auf spielerische und unterhaltende Elemente.

### 9.2.3 Projektbasierte Personas

Die folgenden vier jugendlichen Personas wurden anhand der Beobachtungen der Projekte, die bereits in Kapitel 6.2 vorgestellt wurden, erstellt. Während aller Projekte wurde ein *research diary* geführt, das als Grundlage zur Erstellung der jugendlichen Personas diente. Die Personas haben einen Bezug zu realen Teilnehmern, sie stellen jedoch nicht zwangsläufig genau eine Person dar. Die gewählten Namen sind zur Wahrung der Anonymität der Teilnehmer frei erfunden.

**Djamil, 12 Jahre**

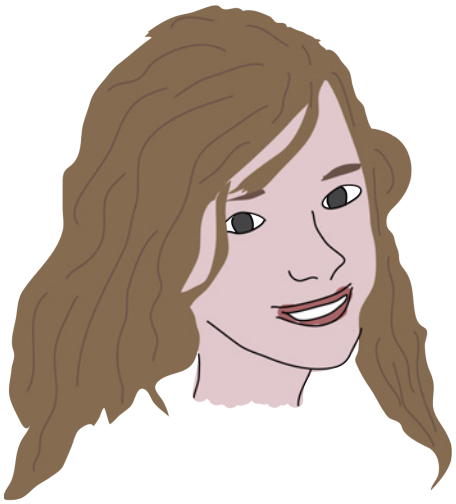
Djamil ist in Afghanistan geboren, seine Familie kam nach Deutschland, als er zwei Jahre alt war. Er spricht akzentfreies Deutsch mit seinen Freunden; mit seiner Familie spricht er sowohl Afghanisch als auch Deutsch. Er ist interessiert an aktueller arabischsprachiger Musik und traditioneller Kleidung und ist gerade dabei, sich Zusammenhänge in diesen Kontexten zu erschließen. Beides versucht er auch seinen vielen Freunden, die keinen arabischen Migrationshintergrund haben, zu vermitteln und über Hintergrundwissen zu informieren.

In der Schule bevorzugt er die Fächer Deutsch und Musik, besonders liegt ihm dort die Interpretation von Texten bzw. Liedtexten. In Mathematik ist er meist zögerlich, da er sich von manchen Aufgaben eingeschüchtert fühlt, obwohl er sie ohne Probleme lösen könnte. Seine Hausaufgaben fertigt er in der Regel zuverlässig an, obwohl es manchmal Aufgaben gibt, die er sehr langweilig findet, da sie keinen Spielraum für eigene Ideen lassen.

Er begreift Computer als eine Möglichkeit, um mit verschiedenen Medien Geschichten zu entwerfen. Sobald er jedoch das Gefühl bekommt, dass das Programmieren zu kompliziert werden könnte, lässt er lieber anderen den Vortritt. Da er zuhause keinen eigenen Computer hat, nutzt er das Computerlabor der Schule in Freistunden und in seiner Freizeit, um z.B. mit *youtube* nach neuer Musik zu suchen. Er hat einen E-Mail-Account, den er häufig nutzt, um seinen Freunden Links zu lustigen Webseiten zu schicken. Darüber hinaus nutzt er manchmal den *instant messenger ICQ*, um zu gucken, wen er von seinen Freunden direkt kontaktieren kann. Er ist nicht bei sozialen Netzwerken angemeldet.

Lehrer beschreiben Djamil als kommunikativ, wenn auch manchmal etwas verträumt. Sie loben seine Fähigkeit, Schüler mit verschiedenen Interessen zusammenzubringen. Negativ erwähnen sie, dass er immer mal wieder seine Unterlagen zuhause vergisst.

**Alina, 12 Jahre**



Alina ist in Deutschland geboren, sie hat einen drei Jahre älteren Halbbruder und eine fünf Jahre jüngere Schwester. Ihre Eltern leben getrennt in weit entfernt gelegenen Stadtteilen. Daher ist es ihr wichtig, z.B. Schulresultate oder Fotos mit ihrer Familie über das Internet teilen zu können. Sie lebt bei ihrer Mutter gemeinsam mit der Schwester. Ihren Vater und den Halbbruder besucht sie mindestens einmal in der Woche und bleibt dann auch über Nacht, da sie ein eigenes kleines Zimmer dort hat.

Mit ihrer besten Freundin möchte sie am liebsten alle Erlebnisse gemeinsam durchleben. Sie träumen davon, einen Roman zu schreiben, der von einer gerechten Welt erzählt, in der alle miteinander gut auskommen. Zuhause bastelt sie gerne an Collagen, die Geschichten über Mädchen in verschiedenen Ländern erzählen. Für ihre kleine Schwester hat sie angefangen, ein Fotoalbum zusammenzustellen, das sie ihr später mal zeigen will.

In der Schule zeigt sie in den meisten Fächern gute Leistungen. Es fällt ihr schwer, ein Lieblingsfach zu benennen, obwohl viele Lehrer vermuten, dass sie besonders den Werkunterricht oder den Kunstunterricht schätzt, da sie in beiden Fächern immer sehr kreative Lösungen umsetzt und auch häufig andere Mitschüler bei der Ideenfindung unterstützt und wertvolle Vorschläge unterbreitet.

Computer sind für sie in erster Linie Kommunikationswerkzeuge, obwohl sie auch vermutet, dass man damit spannende Geschichten erzählen oder tolle Collagen zum Ausdrucken anfertigen kann. Leider spielt ihr Halbbruder viel am Computer, so dass sie ihn nur selten benutzen kann. Zumeist fühlt sie sich aber auch von ihrem älteren Halbbruder eingeschüchtert, der manchmal, wenn auch im Spaß, abfällige Bemerkungen über die von ihr verwendeten Programme macht. Zur Kommunikation verwendet sie ausschließlich E-Mails. Sie hat zwar keinen eigenen *Twitter*-Account, guckt jedoch manchmal nach, was ihr Halbbruder darüber mitteilt. Auf ihrem Nintendo DS nutzt sie häufiger *Art Academy*, bei dem sie ihre Kreationen all ihren Bekannten als Diashow zeigen kann.

Alina wird von ihren Lehrern durchgehend für ihre Kreativität gelobt. Besonders heben sie hervor, dass sie Aufgaben hinterfragt und häufig einen anderen Zugang zu Themen oder Problemen wählt, der nicht selten auch den anderen Teilnehmern als Inspirationsquelle dient. In Mathematik wünschen sich die Lehrer jedoch manchmal eine klarere Strukturiertheit ihrer Lösungsansätze.

**Leon, 13 Jahre**

Leon ist in Deutschland geboren und lebt mit seinen beiden Eltern zusammen. Er hat einen sehr guten Kontakt zu seiner ein Jahr älteren Cousine, die in der französischsprachigen Schweiz lebt, jedoch auch Deutsch sprechen kann. In den Sommerferien besuchen sie sich immer gegenseitig für 10 Tage, um gemeinsam den Wald zu erforschen.

In der Schule hat er viele Freunde, da er in den meisten Fächern sehr gute Leistungen zeigt und seinen Mitschülern bereitwillig seine Hilfe anbietet. Er ist besonders interessiert an mathematischen Fragestellungen und nimmt daher an der Mathematik-Olympiade teil, bei der er stets gut abschneidet. Er geht Probleme strukturiert an und versucht, seinen Lösungsweg nachvollziehbar zu dokumentieren, um damit wieder anderen etwas erklären zu können. In Kleingruppen ist er häufig einer der Wortführer, und seine Aussagen werden von den anderen Teilnehmern akzeptiert, da sie gut nachvollziehbar sind. Nach der Schule besucht er jeweils einmal die Woche den Schulchor und den Klarinettenunterricht der Musikschule.

Er hat einen eigenen Computer, jedoch ist dieser nicht so leistungsstark wie der von seiner Mutter, so dass er es vorzieht, einige Minuten an deren Computer zu verbringen. Seine Mutter entwirft beruflich 3D-Modelle am Computer und zeigt ihm immer mal wieder die Funktionsweise der komplexen 3D-Modellierungs- und Animationsprogramme. Mit der Hilfe seiner Mutter hat er für seine Cousine und sich ein Wiki aufgesetzt, das sie gemeinsam mit Informationen zu Pflanzen und Tieren befüllen. Seine Versuche sich darin auch immer mal wieder auf Französisch auszudrücken, werden von seiner Cousine dann verbessert. Manchmal darf er auf dem iPod Touch seines Vaters spielen, meistens entscheidet er sich dann für Sudoku.

Leons soziale Fähigkeiten werden von den Lehrern hervorgehoben. Sie sehen in ihm einen Schüler, der aktiv ihr Unterrichtsvorhaben unterstützen kann und es schafft, andere Schüler für die Themen zu begeistern. In manchen Situationen haben jedoch einige Lehrer das Gefühl, dass Leon zu viele Interessen gleichzeitig verfolgt.

**Taras, 12 Jahre**

Taras ist in Deutschland geboren, seine Eltern kommen aus der Ukraine, sie sprechen zuhause mit ihm ausschließlich Deutsch. Er spricht daher nur Deutsch, versteht jedoch manche ukrainische Wörter, die er von seinen Großeltern am Telefon hört. Er hat eine zwei Jahre jüngere Schwester und einen fünf Jahre jüngeren Bruder. Er hat seit einem halben Jahr sein eigenes Zimmer und achtet darauf, dass seine Geschwister nicht in sein Zimmer gelangen, wenn er nicht da ist, da er meist einige Lego Technic Modelle dort aufbewahrt und befürchtet, dass sie von seinen Geschwistern zerstört werden könnten.

Zuhause arbeitet er am liebsten zurückgezogen an Themen, die ihn faszinieren; dazu zählen beispielsweise Knobelaufgaben, Schach- und Logikrätsel. Wenn er sich einem dieser Themen annimmt, ist es sehr schwer, mit ihm in Kontakt zu treten. Mit seinem einzigen besten Freund, der ihn manchmal besucht, löst er häufig gemeinsam solche Rätsel; auch hier ist es schwer, die beiden zu kontaktieren oder an ihrem Vorhaben teilzuhaben.

Er hat einen eigenen Computer zuhause, dessen Komponenten er mit einem vier Jahre älteren Bekannten aus der Schach-AG selbst zusammengestellt und -gebaut hat. Er nutzt dort Linux als Betriebssystem, kommt aber auch mit den Windows-Rechnern in der Schule gut zurecht. An den Schulcomputern stören ihn jedoch die fehlenden Admin-Rechte, da er ab und an kleine Zusatzanwendungen ausprobieren möchte, die jedoch nicht von ihm installiert werden können. Er nutzt das *IRC*, um zu bestimmten Themen Informationen von Gleichgesinnten zu erhalten bzw. auszutauschen. Darüber hinaus nutzt er E-Mail und benutzt dafür eine PGP-Verschlüsselung. An seinem Linux-PC zuhause versucht er sich häufig an Shell-Skripten, um einfache Aufgaben zu automatisieren. Zu seinem dreizehnten Geburtstag wünscht er sich

ein Mobiltelefon mit Android-Betriebssystem, um darauf kleine Applikationen zu implementieren.

Die meisten Lehrer sind der Meinung, dass man Taras seine Freiheiten geben muss, weil er dann zumeist sehr gute Leistungen, vor allem in den mathematisch-naturwissenschaftlichen Fächern, abliefert. Schwierigkeiten haben jedoch die Geschichts- und Deutschlehrer, da sie nur schwer einen Zugang zu ihm bekommen und sie nicht einschätzen können, ob er bestimmte Aufgaben nicht versteht oder sie ihn nicht interessieren.

## 9.3 Jugendliche Personas als Gradmesser der Attraktivität

Die aus den Projektbeobachtungen entwickelten jugendliche Personas stellen vier verschiedene Blickwinkel dar, die für die Informatik relevant sind. Diese Relevanz kann verdeutlicht werden, da es Übereinstimmungen zu Rasmussen und Håpnes [1991] gibt, die versucht haben, typische Personengruppen im Informatikstudium zu erkennen und zu beschreiben. So haben Rasmussen und Håpnes vier Personengruppen im Informatik Studium identifiziert: Normale Studierende mit Interessen neben dem Computer (Mehrheit), *Hacker* mit ausschließlichem Fokus auf Technologien (Minderheit), engagierte Studierende mit starker Hingabe für ein erfolgreiches Studium (Minderheit) und weibliche Studierende[1] (Minderheit). Rasmussen und Håpnes legen Wert darauf, dass alle vier Gruppen erkannt und gefördert werden, da sie befürchten, dass in der Informatik aufgrund von prägenden Literaturquellen, wie beispielsweise das Buch von Turkle [1984], ein zu großer Fokus auf *Hacker* gelegt wird und diese so als das prägende Bild wahrgenommen werden. Djamil kann man der Gruppe der normalen, Alina den weiblichen und Leon den engagierten Studierenden zuteilen. Taras repräsentiert die *Hacker*.

Besonders Djamil, Alina und Leon spiegeln auch vielfältige Eigenschaften und Fähigkeiten bei *soft skills* wieder, die in Kapitel 2.2 unter Berufung auf beispielsweise Kabicher et al. [2009] genannt wurden und dort als wichtige Faktoren in der IT-Branche dargestellt wurden.

Nachfolgend kommen alle vier Personas zu Wort: Mit *cognitive walkthroughs* wird beschrieben, welche Aspekte sie von *scratch* im klassischen Gruppenunterricht (ver-

[1] Rasmussen und Håpnes [1991] stellen dabei fest, dass die weiblichen Studierenden selbst als eine eigene Gruppe sehen, obwohl sie sich z.B. der vielen Gemeinsamkeiten mit der Gruppe der normalen Studierenden bewusst sind.

gleiche Kapitel 5.2.3) und dem virtuellen Sandkasten mit den agilen Methoden wahrnehmen und welche Erfolgserlebnisse sie dabei erfahren. Eine Gewichtungsmatrix nach Adlin und Pruitt [2010] erlaubt dann zusätzlich einen Vergleich.

### 9.3.1 Cognitive walkthroughs

Der Kontext der *cognitive walkthroughs* ist Projektarbeit, in der eine gemeinsame digitale Geschichte in einer relativ geringen Zeit zu entwickeln ist. Es soll eine Geschichte erdacht werden, Aufgaben verteilt und Einzelergebnisse wieder zusammengefügt werden. Das finale Ergebnis soll präsentationsfähig und für Außenstehende nachvollziehbar dokumentiert sein.

#### Djamil: *Scratch*, klassischer Gruppenunterricht

Djamil erkennt in *scratch* schnell die Möglichkeit, Musikstücke zu importieren. Ihm fehlt jedoch eine Anleitung, wie man dies umsetzt und wie man diese Musikstücke in das Szenario einbaut bzw. Skripte mit Musikausgabe erstellt. Er traut sich nicht, die anderen Gruppenmitglieder zu fragen, da diese schon weitaus kompliziert wirkende Skripte erstellt haben. Um dies zu überspielen, vertieft er sich darin, immer neue Lieder im Internet probezuhören. Er will einige der Lieder den anderen Teilnehmern vorspielen oder sogar zukommen lassen, diese sind jedoch meist mit anderen Dingen beschäftigt und sehen daher darin eine nicht gewollte Ablenkung. Wenn er vom Lehrer aufgefordert wird, etwas Konstruktives in *scratch* zu vollbringen, verwendet er einen geringen Anteil der Bausteine und ausschließlich diejenigen, die vorher in Einführungskursen oder Tutorien vorgestellt wurden. Auch die dabei entstehenden Skripte sind meist nur leichte Variationen von bereits durch den Lehrer oder Aufgabenblätter vorgestellter Skripte.

#### Djamil: Virtueller Sandkasten, agile Methoden

Durch die morgendlichen *meetings* erfährt Djamil, dass noch weitere Teilnehmer mit ähnlichen Problemen kämpfen, und er weiß nun konkret, wen er zu einem bestimmten Problem befragen kann. Die Arbeit im Paar gefällt ihm gut, er will jedoch einem Partner, der scheinbar gut programmieren kann, nicht zur Last fallen, so dass er sich leicht in einen passiven Part begibt. Die Postergestaltung des *informative workspace* sagt ihm sehr zu, und er nutzt die Gelegenheit, um die Gruppenarbeit zu visualisieren und zu dokumentieren, was wiederum von anderen fortgeschrittenen, programmierenden Teilnehmern wertgeschätzt wird.

Über die Gruppenplattform des virtuellen Sandkastens ist es Djamil möglich, Musikstücke der ganzen Gruppe zugänglich zu machen und zur Diskussion über den Einsatz im Programm zu stellen. Den Gruppenchat verwendet er, um andere auf neue Musik aufmerksam zu machen, aber auch um projektfremde Inhalte zu vermitteln. Er freut sich, wenn im Gruppenchat andere Teilnehmer auf neu erstellte Skripte und deren Funktion hinweisen, da er diese dann gerne ausprobiert und für eigene Zwecke abwandelt. Die abrufbare Autorenhistorie zu jedem Skript findet er in diesem Zusammenhang hilfreich, da er so immer direkt weiß, wen er um Hilfe bitten kann, falls seine Änderungen nicht den erwünschten Effekt hervorrufen. Das Wiki bietet ihm die Gelegenheit, seinen Freunden und Verwandten Inhalte aus der Schule spannend zu präsentieren. Gleichzeitig ist er jedoch eingeschüchtert, da er die wikitypischen Formatierungsbefehle nicht kennt und an mancher Stelle nicht weiß, wie man Seiten und Inhalte verknüpfen kann.

### Alina: *Scratch*, klassischer Gruppenunterricht

Alina findet *scratch* toll, da sie damit Bilder und Figuren zeichnen und malen kann. Mit einfachen Skripten kann sie die selbst gestalteten Kreaturen miteinander sprechen lassen, so dass sie sich Geschichten ausdenken und umsetzen kann. Sie verschickt immer unfertige *scratch*-Projekte per E-Mail an ihre beste Freundin und fordert sie auf, die darin begonnene Geschichte fortzuführen. Beide scheuen sich jedoch, ihre vollendeten Geschichten auf der *scratch*-Webseite zu veröffentlichen, da sie befürchten, ihre programmiertechnische Lösungen wären nicht ausgereift genug, um sie unbekannten Personen zu zeigen. Sobald sie *scratch* in Gruppenarbeit bzw. mit mehr als einer Person verwendet, werden ihr die Prozesse und *scratch*-Programme schnell zu unübersichtlich.

### Alina: Virtueller Sandkasten, agile Methoden

Bei den *meetings* ist Alina immer etwas enttäuscht, dass die anderen Teilnehmer sich auf die programmiertechnische Umsetzung fokussieren und kaum die umgesetzten Story-Inhalte erwähnen. Sie arbeitet sehr gerne im Paar und zieht es vor, wenn die zweite Person etwas fortgeschrittener ist, da sie dann das Gefühl hat, mehr zu lernen. Manchmal kann sie ihren Partnern auch neue Blickwinkel und Erkenntnisgewinne durch unerwartete Fragen eröffnen. Auch das Bearbeiten von Skripten anderer Teilnehmer begreift sie als Lernfortschritt und gleichzeitig als Unterstützung, da sie sich so auf die Inhalte konzentrieren kann und nicht jedes Problem von Grund auf

lösen muss. Die Poster des *informative workspace* findet sie hilfreich, um über ihre inhaltlichen Ideen zu sprechen. Es bereitet ihr großen Spaß, die Poster schön und bunt zu gestalten.

Die Gruppenplattform erleichtert ihr den Einblick in die Teilergebnisse der anderen und erlaubt ihr, inhaltliche Vorschläge hinzuzufügen, um die Geschichten spannender oder lustiger zu gestalten. Dies versucht sie auch im Gruppenchat, indem sie immer wieder neue Storyelemente darin vorschlägt. Für das Wiki wünscht sie sich die Möglichkeit, ähnlich zu den Postern auch eine Onlinecollage zu erstellen, die die Ergebnisse dokumentiert.

### Leon: *Scratch*, klassischer Gruppenunterricht

An *scratch* schätzt Leon den einfachen Einstieg und die vielfältigen Möglichkeiten der vorhandenen Bausteine. Um anderen Gruppenmitgliedern Skripte zu erklären und einen groben Überblick vermitteln zu können, fehlt ihm die Möglichkeit, die Inhalte zu strukturieren. Mit *scratch* entwickelt er Skripte aus vielen Bausteinen und sucht aktiv nach der jeweils elegantesten Lösung eines Problems. Auf der *scratch*-Webseite sucht er manchmal nach Projekten mit ähnlichen Inhalten und untersucht dann, ob die darin verwendeten Skripte bessere Lösungen darstellen als seine eigenen Skripte. Er würde gerne aktiver an der *scratch*-Gemeinschaft teilnehmen, kann dies aber nicht, da die meisten Kommentare und Diskussionen auf der Webseite in Englisch sind und er sich darin noch nicht sicher genug für fachliche Kommentare oder auch Fragen fühlt. Ihm fehlt auch noch ein klarer gemeinsamer Bezugspunkt für Bekannte oder Freunde auf der Webseite; die Möglichkeit Galerien zu erstellen, reicht ihm dazu nicht aus.

### Leon: Virtueller Sandkasten, agile Methoden

Bei den *meetings* interessiert sich Leon in erster Linie für die Probleme der anderen, um ihnen gezielt zu helfen. Im Paar versucht er den jeweiligen Partner so zu unterstützen, dass dieser eigene Erfolge feiern kann. Den *informative workspace* nutzt er gerne als Diskussionsgrundlage. Er findet es gut, wenn andere die Poster erstellen, so dass er sich weiter auf die programmiertechnische Praxis konzentrieren kann. Es bereitet ihm große Freude, wenn er sieht, dass andere Teilnehmer seine Skripte als Grundgerüst verwenden und Änderungen daran vornehmen.

Er nutzt die Gruppenplattform des virtuellen Sandkastens, um verschiedene Medien, gut dokumentiert, für die weitere Nutzung anzubieten. Während der Entwick-

lung versucht er immer, einen Status über sein aktuelles Vorhaben im Gruppenchat mitzuteilen. Er achtet dort auch auf Hilferufe der anderen Teilnehmer. Sollte es sich dabei um allgemeine Probleme handeln, legt er eine Erläuterung auf der Gruppenplattform an, um das Wissen nachhaltig der Gruppe zugänglich zu machen. Er wünscht sich dabei eine Funktion, um bestimmte Themen auch gleichzeitig auf die Wikiseiten zu übertragen. Anhand der Autorenhistorie der Skripte stellt er im Wiki eine Liste aller Autoren der jeweiligen Programme zusammen.

**Taras: *Scratch*, klassischer Gruppenunterricht**

Taras sucht sich in *scratch* gezielt vertrackte Probleme und versucht diese zu lösen. Es kann dabei vorkommen, dass er mehrere Skripte baut, die verschiedene Probleme behandeln, jedoch inhaltlich in keinem Zusammenhang stehen. Er nutzt die Bühne von *scratch* eher als Testumgebung für seine Skripte anstatt einer Präsentationsfläche für Geschichten. Es macht ihm dabei nichts aus, dass andere seine Skripte nicht nachvollziehen können. Auf die Bitte der anderen Teilnehmer oder eines Lehrers hin, macht er anderen Teilnehmern seine Skripte zugänglich und fügt diese in deren Programme ein, jedoch ohne die genaue Herangehensweise des Skripts zu erklären.

**Taras: Virtueller Sandkasten, agile Methoden**

Während der *meetings* ist Taras meist wenig an den Lösungen der anderen interessiert. Er spricht dort auch von seinen Ergebnissen nur wenig und lässt so seine Mitschüler zumeist im Dunkeln. Häufig müssen sehr aufmerksame Mitschüler oder Lehrer auf Anknüpfungspunkte mit den Vorhaben der anderen Teilnehmer hinweisen und eine Kooperation koordinieren. Der Erfolg von *pair programming* hängt bei Taras davon ab, ob er einen weiteren Teilnehmer mit ähnlichen Interessen findet. Dann bildet sich meistens ein Paar, das intern gut funktioniert, jedoch wenig nach außen kommuniziert. Sobald Taras jedoch ein Paar mit jemandem mit anderen Herangehensweisen bilden muss, ist davon auszugehen, dass er die meiste Programmierarbeit alleine durchführt und die zweite Person in eine passive Zuschauerrolle drängt. Den *informative workspace* verwendet er nur nach Aufforderung. Es stört ihn, dass andere Teilnehmer seine Skripte verwenden und dann mit Problemen damit direkt zu ihm kommen.

Er nutzt manchmal die Kommentarfunktion auf der Gruppenplattform des virtuellen Sandkastens, um offene Fragen zu seinen Ergebnissen knapp zu beantworten. Toll findet er die Möglichkeit, die Plattform auch von zuhause nutzen zu können,

da er so auch abends noch Skripte erstellen kann, die er dann am nächsten Tag in der Schule weiter bearbeiten kann. Im Gruppenchat äußert er sich nur, wenn andere besonders vertrackte Probleme beschreiben. Da er den Chat jedoch nur selten auf Neuigkeiten überprüft, antwortet er zumeist mit einiger Verzögerung. Die Wikifunktionalität nimmt er in Kauf, da sie die ihm sonst wenig geschätzte Dokumentationsarbeit ersetzt.

#### Analyse der *cognitive walkthroughs*

Die *cognitive walkthroughs* haben gezeigt, dass Djamil und Alina im klassischen Gruppenunterricht mit *scratch* deutlich weniger soziale Bezugspunkte entdecken können als mit dem virtuellen Sandkasten und den agilen Methoden. Diese Variante erlaubt es beiden, ihr Unsicherheiten gegenüber dem Programmieren in Interaktion mit anderen Mitschülern abzubauen. Für sie wird so deutlich, dass auch bei der Softwareentwicklung soziale Prozesse existieren, dass Arbeiten in Teams Spaß macht und das Programmieren erleichtert. Die beiden Personas verdeutlichen auch, dass diese Aussagen unabhängig vom Geschlecht getroffen werden können, da sich die Betrachtungen hauptsächlich auf Interessen und Herangehensweisen der Personas stützen.

Leon wird in beiden Situationen gut zurecht kommen. Es scheint jedoch ersichtlich, dass seine Mitschüler von der zweiten Variante mit dem virtuellen Sandkasten und den agilen Methoden deutlicher profitieren, da ihn diese Variante in seinen Bemühungen zu helfen und zu erläutern stärker unterstützt.

Taras hingegen würde sich mit *scratch* alleine besser fühlen, da ihn gerade die agilen Methoden zu sehr von der eigentlichen Programmierarbeit abhalten. Trotzdem erscheint es sinnvoll, ihn mit dem virtuellen Sandkasten und den agilen Methoden zu konfrontieren, da es so möglich ist, seine offensichtlichen Schwächen im sozialen Miteinander abzubauen (vergleiche dazu auch Lehmann [1996]).

Erinnert man sich an die Erkenntnisse zu den Systemen aus Kapitel 5.1.3, die auf soziale Interaktion Wert legen, so fällt auf, dass besonders Djamil und Alina von *scratch* nicht ausreichend zur Diskussion angeregt werden und dort keine Bezugspunkte zu einer Gemeinschaft erkennen können, die ihnen helfen könnte, Ängste und Unsicherheiten abzubauen. Der virtuelle Sandkasten mit den agilen Methoden dagegen bietet allen vier Personas eigene Kommunikationswege an, die sie sich entsprechend ihrer Interessen aneignen können.

Die Betrachtungen zu *scratch* im klassischen Gruppenunterricht zeigen allgemein, dass die Referenzen von *scratch* zu den historischen Programmierumgebungen, welche soziale Interaktion fokussieren (vergleiche Kapitel 5.1.3), hauptsächlich technischer Natur sind und noch Potenzial in der Unterstützung von kooperativen Prozessen besteht: *Scratch* vermag es nicht, im Schulkontext eine Gemeinschaft für Lerngruppen entstehen zu lassen. Menschliche Interaktion wird nur über das Forum oder die Webseite gefördert, die selten im Unterricht erreicht werden können.

### 9.3.2 Gewichtungsmatrix

Um einen klaren und detaillierten Überblick über die Eindrücke der einzelnen jugendlichen Personas hinsichtlich der Anforderungen zur Stärkung der sozialen Aspekte aus Kapitel 7 zu erhalten, eignet sich die Gewichtungsmatrix von Adlin und Pruitt [2010] in abgewandelter Form. Die einzelnen eingetragenen Notenwerte stellen dar, welche Attraktivität und Wichtigkeit die einzelnen Personas den eingetragenen Eigenschaften einräumen. Die Notenskala von -1 bis 2 wird wie folgt angepasst und ist ebenfalls bei Göttel [2011b] nachzulesen:

2: Diesen Aspekt erkennt die Persona vollständig und zufriedenstellend wieder.

1: Die Persona glaubt, der Aspekt sei teilweise vorhanden.

0: Dieser Aspekt ist der Persona egal.

-1: Die Persona vermisst diesen Aspekt.

#### Gewichtung

Ein umstrittenes Thema dürfte die Gewichtung sein. Es handelt sich hierbei definitiv nicht um eine Bewertung der Eigenschaften der Personas. Vielmehr wird angenommen, dass bestimmte Jugendliche sich von der IT bereits ausreichend angezogen fühlen, dass sie sich mit hoher Wahrscheinlichkeit für ein Informatik- oder zumindest Informatik-nahes Studium entscheiden werden, unabhängig von Maßnahmen, die die Attraktivität der Informatik steigern sollen. Dies kann man z.B. an Beobachtungen von Potter et al. [2009] festmachen, die zeigen, dass sich gerade Menschen, die bereits in früher Jugend (oder Kindheit) im nicht-schulischen Bereich Zugang zu IT hatten und da bereits eine technische Affinität aufweisen, für eine spätere IT-Karriere entscheiden. Zum Zweiten kommt zum Tragen, dass die stets geforderten *soft skills* (siehe Kapitel 2.2) vielleicht häufiger Personengruppen zuzuschreiben

sind, die nicht so stark an der Informatik interessiert sind, aber entsprechende Fähigkeiten besitzen. In diesem Sinne wird davon ausgegangen, dass besonders Djamil, Alina und Leon von Maßnahmen zu Steigerung der Attraktivität der Schulinformatik angesprochen werden sollten. Sie repräsentieren Personengruppen, die beispielsweise kreative Lösungen entwickeln, Gruppenmitglieder für etwas begeistern und informatische Zusammenhänge nachvollziehen können. Taras dagegen wird mit hoher Wahrscheinlichkeit auch weiterhin gute programmiertechnische Lösungen entwickeln und sich mit besonderen Problemen befassen. Um es noch einmal klar zu stellen: Taras stellt nach wie vor einen wichtigen, wenn auch laut Rasmussen und Håpnes [1991] überproportional präsenten Charakter der Informatik dar. Es geht bei den folgenden Überlegungen nicht darum, die entsprechende Personengruppe zu vernachlässigen. Eine Förderung muss bei solchen Persönlichkeiten jedoch inhaltlich geschehen (z.B. auch auf Ebenen der sozialen Fähigkeiten) und nicht auf der Ebene, die Informatik als Disziplin attraktiv zu vermitteln.

Die tatsächlichen Werte der Gewichtung stellen also nur Vorschläge dar, die jedoch aus den oben genannten *cognitive walkthroughs* nachvollziehbar erscheinen. Djamil und Alina scheinen am meisten davon überzeugt werden zu müssen, dass die Informatik eine attraktive Disziplin darstellt; daher wird ihnen jeweils die höchste Gewichtung mit 35 Punkten zugeteilt. Sie stellen auch die Personengruppen dar, die nach Rasmussen und Håpnes am wenigsten wahrgenommenen sind, aber den Großteil der Studierenden der Informatik stellen und daher einer besonderen Aufmerksamkeit bedürfen, um sie für die Informatik zu gewinnen. Leon dagegen ist schon von der Informatik als solches überzeugt, bei ihm sind beispielsweise interdisziplinäre und soziale Aspekte hervorzuheben, um die bereits wahrgenommene Attraktivität zu erhalten, seine Gewichtung soll 20 Punkte betragen. Ihn könnte man als den engagierten Studierenden nach Rasmussen und Håpnes sehen, der ebenfalls wichtig für die Informatik ist. Bei Taras ist davon auszugehen, dass ihm die Informatik schon als attraktiv genug erscheint und ihn daher Maßnahmen zur Steigerung des Ansehens nicht tangieren. Er soll daher nur eine Gewichtung von 10 Punkten erhalten. Diese Gewichtung der vier Personas kann man auch in Beziehung zu den Betrachtungen von Denning [2002] bringen (vergleiche Kapitel 2.2): So ist leicht nachvollziehbar, dass Djamil und Alina Parallelen zu den dort erwähnten Pragmatikern aufweisen. Maßnahmen zur Steigerung der Attraktivität der Informatik scheinen genau bei dieser großen Personengruppe selten zu fruchten und benötigen besonderen Umgang. Die Gewichtung zugunsten dieser beiden Personas erlaubt weitere Schritte zur Verbesserung des Ansehens der IT deutlicher auszurichten.

| | Djamil | Alina | Leon | Taras | |
|---|---|---|---|---|---|
| Gewichtung | 35 | 35 | 20 | 10 | Gewichtete Summe |
| *Scratch*, klassischer Gruppenunterricht: | | | | | |
| Bezugspunkte Gruppe | -1 | -1 | 1 | 0 | -50 |
| Kollektive Ressourcen | 1 | 1 | 1 | 2 | 110 |
| Gemeinschaftsgefühl | -1 | -1 | 1 | 0 | -50 |
| Wissensaustausch | 1 | 1 | 1 | 2 | 110 |
| Kenntnis Projektfortschritt | -1 | -1 | 1 | 0 | -50 |
| Internes Tutorennetzwerk | -1 | -1 | 1 | 0 | -50 |
| Programmvisualisierung | 0 | 0 | 1 | 1 | 30 |
| Zusätzliche Anreize | -1 | 1 | 1 | 2 | 40 |
| Vielfältigkeit | 1 | 2 | 1 | 0 | 125 |
| Virtueller Sandkasten, agile Methoden: | | | | | |
| Bezugspunkte Gruppe | 2 | 1 | 1 | 0 | 125 |
| Kollektive Ressourcen | 2 | 2 | 2 | 2 | 200 |
| Gemeinschaftsgefühl | 1 | 1 | 2 | 0 | 110 |
| Wissensaustausch | 1 | 2 | 2 | 2 | 165 |
| Kenntnis Projektfortschritt | 1 | 1 | 1 | 0 | 90 |
| Internes Tutorennetzwerk | 1 | 2 | 2 | 0 | 145 |
| Programmvisualisierung | 0 | 0 | 1 | 1 | 30 |
| Zusätzliche Anreize | 1 | 1 | 1 | 2 | 110 |
| Vielfältigkeit | 2 | 2 | 2 | 0 | 180 |

2: Diesen Aspekt erkennt Persona vollständig und zufriedenstellend wieder.

1: Persona glaubt, der Aspekt sei teilweise vorhanden.

0: Dieser Aspekt ist Persona egal.

-1: Persona vermisst diesen Aspekt.

Tabelle 9.3: Gewichtungsmatrix nach Adlin und Pruitt [2010] für die jugendlichen Personas und die Anforderungen zur Stärkung der sozialen Aspekte aus Kapitel 7. Bewertet werden *scratch* im klassischen Gruppenunterricht und der virtuelle Sandkasten mit agilen Methoden.

Für die Gewichtungsmatrix werden die folgenden Eigenschaften geprüft, die bereits in Kapitel 7 identifiziert wurden: Bezugspunkte Gruppe, kollektive Ressourcen, Gemeinschaftsgefühl, Wissensaustausch, Kenntnis Projektfortschritt, internes Tutorennetzwerk, Programmvisualisierung, zusätzliche Anreize und Vielfältigkeit. Die Bewertung ist der Tabelle 9.3 zu entnehmen.

### Analyse Gewichtungsmatrix

Die Gewichtungsmatrix stellt dar, dass *scratch* im klassischen Gruppenunterricht in der Summe den identifizierten vorrangigen Zielgruppen (Djamil und Alina) deutlich weniger attraktiv erscheint als der virtuelle Sandkasten mit den agilen Methoden. Alina würde mit dem virtuellen Sandkasten wahrscheinlich stärker in ihrer Freude am Ausprobieren und der zwischenmenschlichen Interaktion unterstützt werden als Djamil, so dass sie die sozialen Aspekte wahrscheinlich noch etwas deutlicher erkennen könnte als er. Der Matrix kann auch entnommen werden, dass der virtuelle Sandkasten weiteres Potenzial hat, um die identifizierten Anforderungen zur Stärkung der sozialen Aspekte noch deutlicher zu unterstützen bzw. hervorzuheben, da z.B. bei Djamil noch einige Funktionalitäten und Abläufe deutlicher hervorgehoben werden müssen, damit er sie vollkommen und zu seiner Zufriedenheit identifizieren kann. Bei Leon kann man entsprechend der Bewertungen von *scratch* und dem virtuellen Sandkasten davon ausgehen, dass er bereits, wenn auch etwas umständlich, in *scratch* Möglichkeiten sieht, die sozialen Austausch unterstützen. Somit stellen der virtuelle Sandkasten und die agilen Methoden Vereinfachungen für ihn dar, die ihn sicherlich erfreuen, aber nur im geringen Maße einen Einfluss auf sein bereits positives Bild der Informatik ausüben dürften. Taras hingegen dürfte von den beiden unterschiedlichen Varianten kaum verschiedene Meinungen bilden: Er ist stets so sehr auf die technische Ebene und das Programmieren fokussiert, dass er im virtuellen Sandkasten logischerweise die gleichen für ihn entscheidenden Funktionalitäten entdeckt und die agilen Methoden ähnlich der herkömmlichen Gruppenarbeit in Schulen als störenden Nebeneffekt wahrnimmt. Die Benotung von Taras verdeutlicht – beispielsweise bei den Bewertungen zu kollektiven Ressourcen und Wissensaustausch – seine Sonderposition innerhalb der vier Personas. Er wird zusätzliche Angebote zur sozialen Interaktion zwar registrieren, sie werden von ihm jedoch immer als ausreichend wahrgenommen, da er diese nicht für sich in Betracht zieht.

Allgemein wird im Vergleich der gewichteten Summen ersichtlich, dass nahezu alle Anforderungen durch eine Kombination aus virtuellem Sandkasten und agilen

Methoden deutlicher angesprochen werden als mit *scratch* im klassischen Gruppenunterricht. Lediglich bei der Programmvisualisierung ist davon auszugehen, dass hier durch diese Variante keine namhafte Verbesserung auftritt.

### 9.3.3 Einordnung der Bewertungen

Die Blickwinkel der Personas wurden vom Autor eingenommen, um die *cognitive walkthroughs* und die Bewertung in der Gewichtungsmatrix durchzuführen. Es handelt sich somit um eine Argumentation, die Potenzial für weitere Diskussionen und Fragen bereithält und eben nicht quantitativ belegbar ist. Im Rahmen dieser Arbeit erschien solch eine analytische Evaluation (vergleiche Seite 158) als die Lösung, die sowohl den Grad der Validität als auch der Durchführbarkeit sinnvoll in Einklang bringt. In diesem Zusammenhang ist darauf zu verwiesen, dass die Personas nach intensiver Projektarbeit mit insgesamt 75 Schülern entwickelt wurden. So kann davon ausgegangen werden, dass diese Personas und deren Sichtweisen relevante Ergebnisse darstellen. Weitere Überlegungen dazu werden im Ausblick in Kapitel 10.2.1 angestellt.

## 9.4 Abgleich mit Anfangsbetrachtungen

Die Analyse des virtuellen Sandkastens und der angepassten agilen Methoden verdeutlichte, dass die zuvor identifizierten Anforderungen zur Stärkung der sozialen Aspekte der Informatik aus Kapitel 7 durch die Kombination der beiden Maßnahmen abgedeckt werden. Die verschiedenen Personabetrachtungen zeigen unter Berücksichtigung der methodischen Schwächen, dass es die Verwendung des virtuellen Sandkastens und der angepassten agilen Methoden erlaubt, im Schulkontext einen klareren Fokus auf soziale Aspekte der Softwareentwicklung als bisherige Vorgehensweisen und Werkzeuge zu legen. Darüber hinaus kann ein direkter Bezug zur Realität des Berufsbildes hergestellt werden. Soziale Aspekte und Realitätsbezug wurden bereits in Kapitel 2 im Zusammenhang mit dem Selbstbildnis der IT und den Anforderungen an Arbeitnehmer in der IT-Branche als wichtige unterrepräsentierte Faktoren für die Attraktivität der IT identifiziert.

Auch die Kapitel 2, 4 und 5 zeichneten ein ähnliches Bild für die Informatikbildung, die CCI und die ILEs: Schülern mangelt es bisher an Möglichkeiten, dynamische soziale Prozesse und Kooperation im Zusammenhang mit IT zu erleben und folglich auch zu erkennen. Praktisch konnte dies in Kapitel 6 anhand der Pro-

jektbeobachtungen festgestellt werden. Dementsprechend erscheint die Kombination aus virtuellem Sandkasten und den angepassten agilen Methoden als ein vielversprechender Weg zu einem angemessenen Ansehen der Informatik unter Jugendlichen. Gerade die Kombination aus technischen Eigenschaften, die soziale Interaktion fördern und didaktisch-methodischen Maßnahmen zur Strukturierung der Gruppenarbeit mit Realitätsbezug entsprechen den Gedanken des *blended learning* und sind so ohne weiteren Aufwand in Unterrichtssituationen einsetzbar. Durch die Vielzahl an heterogenen Projekten aus denen heraus der Prototyp und die Anpassungen entstanden sind, ist davon auszugehen, dass nur wenige Änderungen durchzuführen sind, um sie in der Breite im Schulkontext einsetzten zu können. Natürlich sollten diese nicht die einzigen Vorgehensweisen darstellen, um die Attraktivität der Informatik der Realität entsprechend zu kommunizieren. Sie erscheinen jedoch als bemerkenswert, da sie besonders unterrepräsentierten Personengruppen des IT-Arbeitsmarkts zugute kommen bzw. diese berücksichtigen.

## 9.5 Kapitelzusammenfassung

In drei Schritten wurde in diesem Kapitel der virtuelle Sandkasten mit den agilen Methoden auf die Abdeckung der Anforderungen zur Stärkung der sozialen Aspekte aus Kapitel 7 hin untersucht. Zunächst wurde mittels einer Tabelle geprüft, ob und welche Eigenschaften des virtuellen Sandkastens und der agilen Methoden die einzelnen Anforderungen theoretisch abdecken. Damit konnte gezeigt werden, dass je Anforderung mindestens eine Funktionalität oder auch agile Methode existiert, die darauf einwirken kann.

Für den zweiten und dritten Schritt musste zunächst das Personakonzept prinzipiell vorgestellt werden. Daraufhin wurden jugendliche Personas präsentiert, die verschiedene Personengruppen im Schulkontext darstellen und auf Projektbeobachtungen aus Kapitel 6.2 beruhen. Die vier Personas Djamil, Alina, Leon und Taras wurden dann verwendet, um *cognitive walkthroughs* für *scratch* im klassischen Gruppenunterricht und für den virtuellen Sandkasten mit angepassten agilen Methoden zu erstellen. Diese *cognitive walkthroughs* haben verdeutlicht, dass gerade Djamil und Alina von der Variante mit dem virtuellen Sandkasten und den angepassten agilen Methoden profitieren dürften.

In einem letzten Schritt wurde eine Gewichtungsmatrix für die Personas und die Anforderungen aus Kapitel 7 angelegt. Die Matrix war ebenfalls in die beiden Varianten (*scratch* im klassischen Gruppenunterricht und virtueller Sandkasten

mit angepassten agilen Methoden) unterteilt, um die Unterschiede dazwischen zu verdeutlichen. Auch hier ergab sich, dass besonders Personengruppen, die an der Informatik zweifeln, mit der Variante virtueller Sandkasten und agile Methoden die sozialen Aspekte bzw. die zugehörigen Anforderungen besser identifizieren können.

# Kapitel 10

# Zusammenfassung und Ausblick

Dieses letzte Kapitel beschreibt den Ablauf der vorliegenden Arbeit, ordnet die Ergebnisse ein und gibt einen Ausblick. Hierbei werden konkrete, weitere Veränderungen am Prototyp und an den agilen Methoden vorgeschlagen, aber auch allgemein weitere Schritte zur Verbesserung des Ansehens der Informatik empfohlen.

## 10.1 Zusammenfassung der Arbeit

Die vorliegende Arbeit begann mit allgemeinen Betrachtungen zum Spannungsfeld Informatik und Gesellschaft. Es konnte anhand von Literaturquellen gezeigt werden, dass Jugendliche ein einseitig negatives Bild der Informatik vor Augen haben. Sie sehen darin hauptsächlich eine Disziplin für Solisten. Darüber hinaus konnte verdeutlicht werden, dass diese Wahrnehmung dem Selbstbild der Informatik widerspricht, gleichzeitig jedoch zu wenig eindeutige Bekenntnisse zu sozialen Prozessen in der Informatik getroffen werden. Während nach Literaturlage auch das allgemeine gesellschaftliche Ansehen der Informatik negativ geprägt ist, gibt es aussichtsreiche Erkenntnisse für zukünftige Maßnahmen zur Anpassung des Images der Informatik: Viele Untersuchungen zu Frauen in der Informatik haben ergeben, dass diese die Informatik deutlich attraktiver wahrnehmen, wenn soziale Aspekte in der Informatikbildung und in verwendeten Systemen herausgehoben werden.

Will man ein fest sitzendes Image verändern, so sollte dies möglichst frühzeitig und in einem Bildungskontext geschehen. Aus diesem Grund stellt diese Arbeit Literatur zu agilen Methoden in der Bildung vor und kommt dabei zu dem Schluss, dass diese Herangehensweisen nur im universitären Rahmen zu entdecken sind und daher zu spät eingesetzt werden, um nachhaltigen Einfluss auf das Ansehen der Informa-

tik unter Jugendlichen nehmen zu können. In den Betrachtungen zur Informatikbildung im Schulkontext konnte festgestellt werden, dass Gruppen- und Projektarbeit an Schulen im Informatikunterricht verbreitet sind, diese jedoch nicht ausreichend strukturiert sind, um soziale Prozesse in den Vordergrund zu stellen. Des Weiteren werden im Unterricht zu wenig Bezüge zur realen Softwareentwicklung hergestellt, um den Schülern ein wirklichkeitsnahes Bild vermitteln zu können.

Neben dem Schulkontext gibt es ein weiteres Forschungsfeld, das sich mit Kindern und Jugendlichen als Nutzer von Computersystemen auseinandersetzt – die *Child Computer Interaction*. Auch dort ist zu vermuten, dass jugendliche Nutzer solcher Systeme Rückschlüsse auf das Wesen der Informatik ziehen. Dies machte es nötig, die *Child Computer Interaction* vorzustellen und herauszuarbeiten, dass auch hier derzeit nur selten soziale Prozesse berücksichtigt werden bzw. aktiv gefördert werden. Dies ist besonders verwunderlich, da erfolgreiche ältere Arbeiten der *Child Computer Interaction* besonders soziale Lernumgebungen propagiert und wichtige Eigenschaften von Systemen beschrieben haben.

Aus der *Child Computer Interaction* heraus sind viele Entwicklungsumgebungen für Novizen bzw. Jugendliche entstanden. Nach einem geschichtlichen Abriss wurden die kaum noch verwendeten Systeme vorgestellt, welche soziale Aspekte deutlich fokussieren und die *initial learning environments scratch* und *Greenfoot*, die aktuell die weiteste Verbreitung besitzen. Anhand der Beschreibungen zu *scratch* und *Greenfoot* wurde klar, dass diese besonders gut funktionieren, wenn eine Person ein eigenes Projekt mit dem jeweiligen System umsetzen möchte.

Sämtliche Betrachtungen stützten sich bis dahin auf Literaturquellen und vereinzelte Stichproben bei Studierenden oder auch Lehrern. Im nächsten Schritt sollte in heterogenen kurzzeitigen Projekten untersucht werden, wie jugendliche Kleingruppen mit gemeinsamen Zielen mit *scratch* unter Verwendung bestimmter agiler Methoden kooperieren. Dabei kam zum Vorschein, dass über alle Projekte hinweg Probleme bei der Kommunikation und dem Austausch von Medien auftraten. In einem Projekt mit *Greenfoot* konnte festgestellt werden, dass die Probleme auch mit dieser Umgebung auftreten, so dass man davon ausgehen kann, dass diese Beobachtungen auch auf *Greenfoot* übertragbar sind.

Aus den Beobachtungen wurden Anforderungen zur Fokussierung auf soziale Aspekte der Softwareentwicklung im Schulkontext herausgearbeitet. Diese Anforderungen sind entsprechend den Gedanken des *blended learning* für Systeme und didaktisch-methodische Herangehensweisen zu verstehen. Konkret konnten neun Anforderungen genannt und in die drei folgenden Kategorien eingeteilt werden:

- Austauschmöglichkeiten anbieten
- *awareness* schaffen
- Außendarstellung fördern

Im nächsten Schritt wurden der im Rahmen der Arbeit entstandene virtuelle Sandkasten und die angepassten agilen Methoden vorgestellt, die zusammen genommen den aufgestellten Anforderungen zur Stärkung der sozialen Aspekte gerecht werden sollen. Beim virtuellen Sandkasten handelt es sich um eine Erweiterung von *scratch*, die einen Fokus auf die soziale Interaktion innerhalb der Gruppe legen soll. Im Wesentlichen werden eine Gruppenplattform, ein Gruppenchat und ein Wiki integriert, um Schüler direkt in der gewohnten *scratch*-Umgebung über Aktivitäten der anderen Teilnehmer zu unterrichten und zur Kommunikation anzuregen. Die Anpassungen bei den agilen Methoden berücksichtigen besonders den Schulkontext und jugendliche Eigenschaften, die schnell dazu führen können, dass der Fokus auf das gemeinsame Ziel verloren geht.

Der virtuelle Sandkasten und die angepassten agilen Methoden wurden dann in drei Schritten theoretisch auf ihre Fähigkeiten geprüft, soziale Aspekte deutlich hervorzuheben. Im ersten Schritt wurden die Anforderungen zur Steigerung der sozialen Aspekte der Informatik mit dem virtuellen Sandkasten und den agilen Methoden abgeglichen. Dies zeigte, dass theoretisch alle Anforderungen abgedeckt werden können. Im zweiten Schritt wurden die jugendlichen Personas Djamil, Alina, Leon und Taras vorgestellt. Diese halfen dabei, *cognitive walkthroughs* aus deren Sichtweisen für *scratch* in klassischer Gruppenarbeit und den virtuellen Sandkasten mit den agilen Methoden anzufertigen. Dies förderte zu Tage, dass der virtuelle Sandkasten und die agilen Methoden gerade musische und kreative Personen wie Djamil, Alina und auch besonders hilfsbereite und wissbegierige Personen wie Leon von den sozialen Aspekten der Softwareentwicklung überzeugen können. In einem letzten Schritt wurde eine Gewichtungsmatrix für die jugendlichen Personas aufgestellt. Diese zeigte deutlich, dass gerade die eher an der Informatik zweifelnden Personas besonders vom virtuellen Sandkasten und den agilen Methoden profitieren und so Informatik als eine Disziplin wahrnehmen können, die auf sozialer Interaktion aufbaut. Wenngleich die Evaluationsschritte formal theoretisch unter Verwendung etablierter Werkzeuge des *Usability Engineering* vorgenommen wurden, so ist diese Ausarbeitung gestützt auf Praxiserkenntnisse aus acht Projekten im Schulkontext mit insgesamt 75 Schülern. Vor allem die ausgewählten Personas tragen dazu bei, das Potenzial und den positiven Einfluss der gewählten Maßnahmen auf das Image der

Informatik zu veranschaulichen: Die präsentierten Maßnahmen erlauben im Schulkontext real verwendete Herangehensweisen der Softwareentwicklung mit sozialem Charakter zu vermitteln. Die bewusste Kombination aus technischen Unterstützungen und methodisch-didaktischen Strukturen ermöglicht eine schnelle und einfache Adaption im Unterrichtskontext.

### 10.1.1 Einordnung der Ergebnisse

Es ist anzumerken, dass die Projekte mit den Schülern stets vom Autor dieser Arbeit betreut wurden und die Jugendlichen über dieses Vorhaben in Kenntnis gesetzt wurden. So waren bei anschließenden Fragebögen oder Befragungen zur Steigerung der Attraktivität der Informatik wohlwollende, positive Antworten und somit verfälschte Aussagen gemäß des Pygmalion-Effekts (vergleiche [Rosenthal und Jacobson, 1968; Rosenthal, 1995]) zu erwarten, die daher nicht für die Arbeit zu berücksichtigen wären. Diese Überlegungen führten zum Einsatz etablierter Methoden aus dem *Usability Engineering* (Personas, *cognitive walkthroughs*, Gewichtungsmatrix), die zunächst theoretischer Natur sind. Die Qualität und Verwertbarkeit der damit getroffenen Aussagen hängen maßgeblich von der zugrunde liegenden Wissensbasis ab. In der vorliegenden Arbeit wurde mittels einer umfangreichen Literaturrecherche zu sozialen Aspekten in der Informatikbildung ein theoretischer Hintergrund erarbeitet. Besonders relevant für eine gültige Herleitung der theoretischen Evaluation sind die praktischen Kenntnisse: Auf der Grundlage weitreichender Erfahrungen aus acht Projekten mit insgesamt 75 Schülern erfolgte eine umfassende Analyse, die Chancen und potenzielle Herausforderungen dieses Ansatzes somit praktisch valide identifiziert.

Im Ergebnis stellt diese Arbeit erstmals einen weitreichenden Zusammenhang zwischen dem Nachwuchsproblem der Informatik und der Vernachlässigung sozialer Aspekte im Informatikunterricht her. Davon ausgehend wird in dieser Arbeit ein Lösungskonzept präsentiert, das aus einer Softwarekomponente und entsprechend angepassten Arbeitsmethoden besteht. Daran wird gezeigt, dass die Einbettung von sozialen, im IT-Alltag etablierten Arbeitsprozessen großes Potenzial hat, informatikinteressierte, aber zögerliche Schüler geeignet an die Informatik heranzuführen. Dabei muss beachtet werden, dass Jugendliche ihr eigenes Sozialverhalten noch erproben. Die Erkenntnisse aus dieser Arbeit erlauben es nun auch Jugendlichen in der Informatikbildung Raum für ihre soziale Entwicklung zu geben und so die Attraktivität der Informatik maßgeblich zu steigern. Somit bietet die Arbeit einen konkreten

Ansatz, den Nachwuchssorgen der Informatik zu begegnen und entsprechende Langzeitstudien zu konzipieren (siehe Ausblick).

Gleichzeitig lässt sich hervorheben, dass nach den Befürchtungen einer ausschließlichen Fokussierung auf große und aufwändige Projekte in der *Child Computer Interaction* (Kapitel 4.1.2) diese Arbeit auf Beobachtungen heterogener und kurzlebiger Projekte basiert. Daher sind die daraus entstandenen Anforderungen an *initial learning environments* zur Unterstützung von jugendlichen Gruppen leicht zu prüfen und so auch für Lehrer und Wissenschaftler praktikabel, die nicht in groß angelegte Forschungsstudien involviert sind. Vor allem stellen die angepassten agilen Methoden didaktisch-methodische Maßnahmen bereit, die ebenfalls von diesen Personengruppen ohne großen Zeitaufwand umsetzbar sind und die so einen Bezug zur modernen Softwareentwicklung herstellen können. Der virtuelle Sandkasten stellt einen Prototyp dar, der die Möglichkeiten zur technischen Unterstützung von Gruppenprozessen deutlich macht und dementsprechend auch im Unterricht eingesetzt werden könnte. Hierbei muss erwähnt werden, dass es bereits eine neuere Version von *scratch* gibt und eine Version 2.0[1] angekündigt ist, die nicht mehr auf *Squeak* aufbauen wird. Daher ist anzudenken, die Anpassungen des virtuellen Sandkastens dann in die neue Version von *scratch* zu übertragen, da in der Regel bei Schülern kein Verständnis für die Verwendung älterer Versionen vorhanden ist.

## 10.2 Ausblick

### 10.2.1 Implementation

Während diese Arbeit hauptsächlich argumentiert, dass soziale Aspekte der Softwareentwicklung hervorgehoben werden sollen, um die Attraktivität der Informatik zu steigern, ist es gleichzeitig auch ein Ziel der vorgestellten Implementation und methodisch-didaktischen Vorgehensweisen, die sozialen Fähigkeiten der Schüler zu stärken. Unter diesem Gesichtspunkt wäre es in Anlehnung an McKinney und Denton [2005] (Kapitel 3.1.1) wünschenswert, jedem einzelnen Teilnehmer konstantes Feedback über das Verhalten in der Gruppe und die Lernerfolge geben zu können. Darin läge nämlich das Potenzial, auf weitere soziale Komponenten einzugehen, die mit den aktuellen Lösungsvorschlägen nur nebenher bzw. implizit erreicht werden: Kritikfähigkeit und die Fähigkeit, konstruktive Kritik zu äußern.

[1] *scratch* 1.4 ist die letzte Version vor dem Versionssprung auf 2.0.

### Der virtuelle Sandkasten

Für den virtuellen Sandkasten sind einige Erweiterungen denkbar, die das Arbeiten damit in Kleingruppen weiter verbessern könnten und auch die Gebrauchstauglichkeit steigern.

So sieht die aktuelle Implementation beispielsweise vor, sämtliche Medien und *scratch*-Programme auf einer Gruppenplattform (hier: CommSy) zu speichern und zu kommentieren. Wünschenswert wäre hier die direkte Einbindung in *scratch*, wie sie auch bei der Möglichkeit zum Hochladen der Ergebnisse auf die *scratch*-Webseite gegeben ist. Auch wäre eine direkte Kontextauswahl beim Start von *scratch* bzw. dem Erstellen oder Öffnen eines *scratch*-Programms hilfreich, da dann automatisch sämtliche Zwischenspeicherungen im Projektkontext der entsprechenden Plattform stattfänden. Somit entfiele eine Speicherung in Ordnerstrukturen auf physischen Medien. Es zeigte sich nämlich, dass es nicht ausreicht, eine externe Plattform anzubieten, die dies ermöglicht. Somit würde ein Zwang entstehen, projektbezogen Medien und Programme abzuspeichern. Da dies durchaus eine zu hinterfragende Praxis darstellt, empfiehlt sich eine Evaluation, die sich besonders auf die Akzeptanz der jugendlichen Nutzer konzentriert.

Eine andere Erweiterung, deren Akzeptanz in Studien mit jugendlichen Nutzern untersucht werden sollte, wäre ein in den virtuellen Sandkasten integriertes Editorfenster für die Wiki-Funktionalität. Diese Variante sollte dann in Nutzertests verglichen werden mit dem aktuell implementierten forcierten Kontextwechsel bei Betätigen des Wiki-Knopfs.

Die verwendeten Onlineplattformen bieten aktuell keine Möglichkeit, um anonyme Beiträge oder Kommentare zu erstellen. Dies ist zwar nicht immer wünschenswert oder sinnvoll im Schulkontext, trotzdem sollte es gerade in den ersten Phasen der Projektarbeit angeboten werden, um die beobachteten Unsicherheiten der Schüler beispielsweise beim Brainstorming zu minimieren.

Es stellt sich auch die Frage, ob das CommSy langfristig den Anforderungen des virtuellen Sandkastens genügt, der auf eine deutlich jüngere Zielgruppe als das ursprüngliche CommSy ausgelegt ist (trotz der positiven Beobachtungen von Otto [2002] im Schulkontext). Auch hier wären andere häufig empfohlene Systeme, z.B. *moodle* (vergleiche Homberg [2006]), zu untersuchen und einzubinden. Gegebenenfalls würde sich auch eine Anpassung der jeweiligen Gruppenplattform-Oberfläche an die Ästhetik von *scratch* anbieten, um den Zusammenhang noch deutlicher zu machen. Auch hier wären wieder Nutzerstudien angebracht, um eine Plattform mit

sinnvoller Oberflächengestaltung zu identifizieren oder gegebenenfalls dahingehend zu erweitern. Zusätzlich wäre strukturell zu hinterfragen, ob solch eine Plattform überhaupt nötig ist oder ob eine Kombination aus *cloud* und *social media* (vergleiche Baumgartner und Himpsl [2008]) einen ähnlichen Funktionsumfang leichtgewichtiger und somit nutzernah bzw. nutzerfreundlich anbieten könnte (siehe Abbildung 10.1). Diese Überlegungen scheinen auch angebracht, da laut der Entwickler ab *scratch* 2.0 auf eine webbasierte Lösung mit Flash gesetzt wird. Es ist fraglich, ob es davon eine quelloffene Version geben wird oder eine API angeboten wird, um *scratch* dann an andere Anwendungen anbinden zu können.

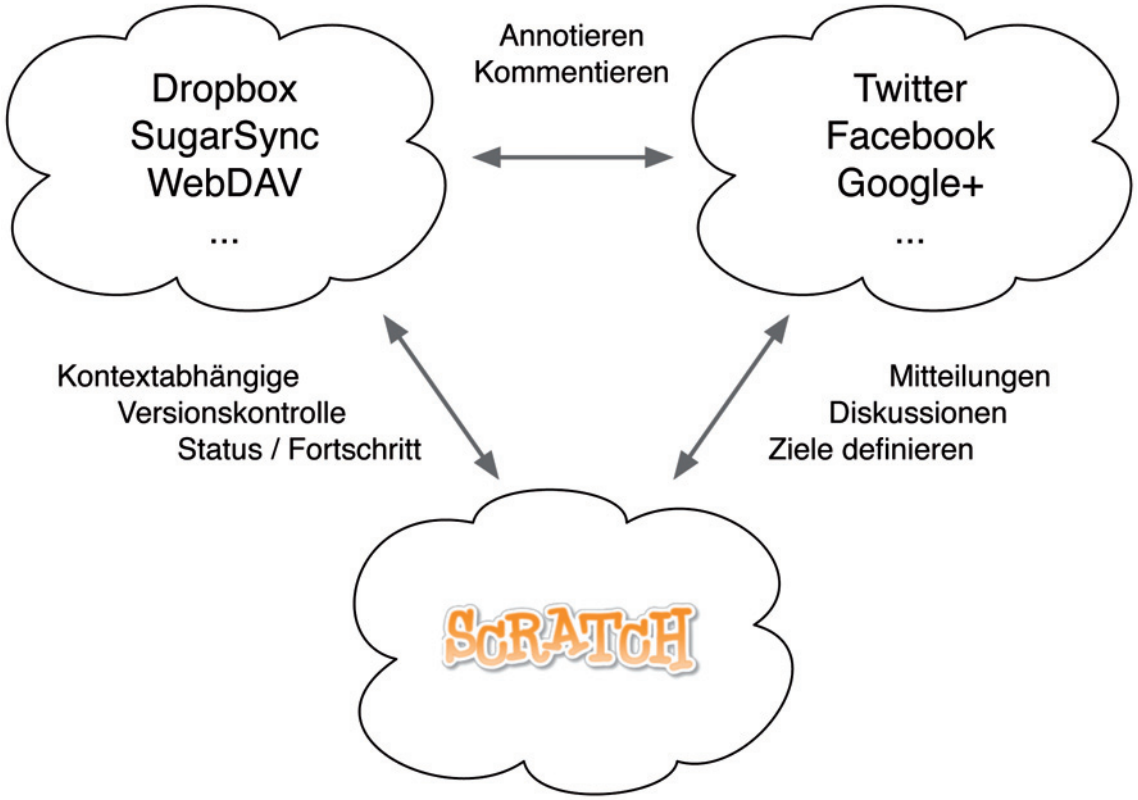



Abbildung 10.1: Mögliche Struktur für *scratch* bzw. den virtuellen Sandkasten, die auf *cloud*-Technologien und *social media* zurückgreift.

Eine Unannehmlichkeit stellt die gemeinsame *Codebasis* dar. Momentan ist es sowohl mit *scratch* als auch mit dem virtuellen Sandkasten umständlich, Projekte und Skripte zusammenzuführen bzw. darüber an einer gemeinsamen *Codebasis* zu arbeiten. Hierfür sollte eine Infrastruktur bzw. eine interne Verwaltung von *scratch*-Projekten entwickelt werden, die es ermöglicht, dass verteilt auf ein gemeinsames Projekt zugegriffen werden kann. Ein offenes Problem wäre bei solch einer Lösung

nach wie vor die Versionierung der einzelnen Änderungen. Naheliegend dabei wäre es, auf professionelle Werkzeuge zur Versionsverwaltung bzw. *concurrent versions systems* (CVS) wie *Apache Subversion* oder *Git* zurückzugreifen. Den Aussagen von Fisker et al. [2008] zufolge ist dies jedoch im Schulkontext äußerst fraglich, da sie beispielsweise selbst in elementaren Informatikkursen an Universitäten den Einsatz von professionellen CVS als zu komplex ansehen. Bevor solche Werkzeuge in den virtuellen Sandkasten integriert werden, wäre es empfehlenswert zunächst eine getrennte Betrachtung anzustellen, wie jugendliche Nutzer eine Versionsverwaltung interpretieren und wie man entsprechende Funktionalitäten für diesen Anwendungskontext nachvollziehbar entwickelt.

Die Programmvisualisierung ist prinzipiell technisch umsetzbar. Erste Implementationen sind im Rahmen dieser Arbeit entstanden, um Skripte vom virtuellen Sandkasten als Bilddatei direkt an eine Online-Bildbearbeitung (*pixlr*[2]) zu schicken, um dort Jugendlichen manuell zu erlauben, Abhängigkeiten zu kennzeichnen. Schnell wurde dabei ersichtlich, dass dieser Export und auch die Bearbeitung der Bilddateien mit Schülern partizipativ gestaltet werden muss, damit so ein Arbeitsprozess nutzbar und verständlich ist. Hier ist demnach ein zusätzlicher Anknüpfungspunkt für weitere Forschungsvorhaben.

### Die agilen Methoden

Eine sicherlich sehr kommunikative Komponente der agilen Methoden erschien in Projekten mit einer maximalen Dauer von fünf Tagen nicht umsetzbar bzw. nicht fair: der Kunde bzw. Auftraggeber vor Ort (*customer on site*). Dieser prüft ursprüngliche Ziele, verfeinert sie und/oder definiert sie neu und tauscht sich so ständig mit den Entwicklern aus. In einer Einbettung der agilen Methoden über einen längeren Zeitraum bzw. für ein längeres Projekt über ein Schuljahr wäre diese Komponente jedoch durchaus sinnvoll, da dort rollenspielartig Probleme und Missverständnisse diskutiert, konstruktive Kritik geäußert und entsprechend Lösungswege neu ausgerichtet werden müssen. Es ist davon auszugehen, dass gerade die *user stories* von dieser Praktik profitieren würden und man so einen noch klareren Bezug zum Berufsfeld schaffen könnte. Es wäre auch zu erarbeiten, ob ein Lehrer bzw. Dozent den Kunden darstellt oder ob dafür Schüler eingesetzt werden könnten.

Ein weiteres offenes Problem der agilen Methoden ist die Leistungsbewertung. Diese Arbeit musste sich damit nicht befassen, da es sich ausschließlich um freiwil-

[2]http://pixlr.com/editor/, zuletzt besucht am 15. April 2012.

lige Projekte ohne Notenrelevanz handelte. Trotz allem sei darauf verwiesen, dass möglicherweise auch hier ein Bezug zu einer gängigen Praxis in der modernen Softwareentwicklung hergestellt werden kann. Dort wird nämlich häufig auf fundierte Selbstbeurteilungen bzw. Selbsteinschätzungen und anschließende *peer reviews* dieser Dokumente durch die Gruppenmitglieder gesetzt. So kann sichergestellt werden, dass die eigenen Einschätzungen zur Geltung kommen und nur die akzeptiert werden, die mit dem Eindruck der anderen Teilnehmer übereinstimmen. Sicherlich handelt es sich bei der schulischen Notenvergabe um einen anderen Kontext, so dass davon auszugehen ist, dass hier methodisch-didaktische Anpassungen erfolgen werden.

Unter der Voraussetzung einer Einbettung über einen längeren Zeitraum, z.B. eines gesamten Schulhalbjahrs, ist es mit Sicherheit auch wünschenswert, die programmiertechnischen Praktiken der agilen Methoden nach Weigend [2005] in den Informatikunterricht einzubetten.

### Experimentendesign und Evaluation

Es ist durchaus denkbar, unter Berücksichtigung der oben genannten Schwierigkeiten (Abschnitt 10.1.1), eine Langzeitstudie zu entwerfen, die eine mögliche Steigerung der Attraktivität der Informatik durch den virtuellen Sandkasten und die angepassten agilen Methoden als Betrachtungsgegenstand untersucht. Dabei müsste man interdisziplinär (mindestens Softwareentwickler, Pädagogen, Lehrer und Psychologen) und mit mehreren Versuchsgruppen (mit Kontrollgruppe) arbeiten und einen Rahmen schaffen, der es erlaubt, über mehrere Schuljahre entsprechende Methoden und Werkzeuge anzuwenden. Darüber hinaus müsste eine Möglichkeit geschaffen werden, die Biografien der Teilnehmer über Jahre hinaus auch nach der Schullaufbahn verfolgen zu können. Die Identifikation der unabhängigen und abhängigen Variablen ist in so einem Vorhaben eine ernsthafte Hürde, so dass darauf gesondert durch Pädagogen und Psychologen eingegangen werden müsste. Folglich ist davon auszugehen, dass Pädagogen und Psychologen federführend bei einer entsprechenden Konzeption beteiligt sein müssen.

### Einbettung in den Schulkontext

Während durchaus die Möglichkeit gegeben ist, dass Lehrer ihre zukünftige Unterrichtsgestaltung auf die vorliegende Arbeit (bzw. daraus entstandene Publikationen) stützen, ist davon auszugehen, dass es sich dabei um Einzelfälle handeln wird. Daher existiert eine weitere wichtige Aufgabe, die interdisziplinär gelöst werden muss.

Sie betrifft die Einbettung der vorgestellten Maßnahmen in den Schulkontext. Hier gilt es, in einem ersten Schritt Pilotprojekte zu installieren, die von Angehörigen der Schulministerien, Schulen und Universitäten getragen werden und geeignete Themen der Informatikrahmenpläne aufgreifen und projektartig – angelehnt an die moderne Softwareentwicklung – umsetzen. Es ist davon auszugehen, dass solch ein Anliegen eine höhere Erfolgsaussicht besitzt, wenn Langzeitstudien (siehe oben) positive Argumente lieferten und auch begleitende Evaluationen sichergestellt wären. Andererseits ist zu bedenken, dass häufig über den Fachkräftemangel in der IT-Branche geklagt wird, so dass eher Handeln gefragt ist, als langwierige Vorstudien durchzuführen.

Zusätzlich sei gesagt, dass gerade Praktiken wie das *pair programming* zunächst im Widerspruch zu Offensiven wie *Schulen ans Netz e.V.*[3] oder *Das macht Schule*[4] zu stehen scheinen, die sich verständlicherweise dafür stark machen bzw. den Eindruck vermitteln, dass möglichst jedem Schüler ein PC zur Verfügung stehen sollte. Es soll hierbei jedoch nicht darum gehen, diese Forderungen zu entkräften. Vielmehr verstärkt diese Bewegung das Bewusstsein, dass über die agilen Methoden hinaus *initial learning environments* entwickelt bzw. angepasst werden müssen, die ein gemeinschaftliches Arbeiten an einem oder mehreren Rechnern attraktiv zu vermitteln wissen. Nur auf solche Weise könnten Lehrer und Dozenten überzeugend z.B. auch *pair programming* einsetzen, da sie so klare fachliche Argumente und professionelle Referenzen für den Einsatz vom *pair programming* und weiteren Praktiken der agilen Methoden vorbringen könnten.

### 10.2.2 Weitere Potenziale zur Steigerung der Attraktivität

Als weitere wichtige Maßnahmen zur Steigerung der Attraktivität sind kreative Elemente der Informatik im schulischen Bereich hervorzuheben und zu vermitteln, da dies für Jugendliche entscheidende Aspekte bei der Berufswahl sind [Romeike, 2007b]. In der Softwareentwicklung werden von Romeike [2008] vier Punkte ausgemacht, die kreative Aspekte darstellen, aber noch zu selten im Schulkontext zu finden sind. Diese seien im Folgenden kurz genannt, um einen genaueren Eindruck zu erhalten, wo in diesem Themenfeld Potenzial zur Verbesserung des Ansehens der Informatik zu sehen ist (vgl. [Romeike, 2008]):

[3] http://www.schulen-ans-netz.de, zuletzt besucht am 15. April 2012.
[4] http://www.das-macht-schule.net, zuletzt besucht am 15. April 2012.

- *Findung/Bestimmung einer Challenge*: Kreativität ist gefragt, wenn die Problemlage anhand von Anforderungen analysiert werden muss und verschiedene mögliche Lösungen in Betracht gezogen bzw. ausprobiert werden müssen.

- *Problemmanagement*: Als hochgradig kreativ wird angesehen, dass Herangehensweisen in der Softwareentwicklung immer wieder hinterfragt werden müssen und auf Unvorhergesehenes eingegangen werden muss. Auch der Umgang mit Programmierfehlern und die Reaktionen darauf sind als kreative Handlungen zu sehen.

- *Produkt*: Der Umgang mit einem fertig gestellten Produkt und möglichen Erweiterungen und/oder Verbesserungen ist als kreativer Aspekt zu behandeln und zu fördern.

- *Präsentation*: Die Darstellung eines fertigen Softwareprodukts in Präsentationen erlaubt kreativen Umgang mit dem Entstandenen.

Ein positiver Nebeneffekt dürfte dabei auch sein, dass Schülern auf diesem Weg der Zugang zu kreativen Lösungswegen erleichtert wird und so davon auszugehen ist, dass sie allgemein besser auf die *Creative Society* nach Resnick [2007] vorbereitet sind, in der er davon ausgeht, dass Kreativität eine der Kernkompetenzen – unabhängig vom IT-Wissen – der Zukunft ist.

Gleichzeitig ist es jedoch auch in Informatik-Studiengängen selbst wichtig, durch geeignete Methoden kreative Aspekte kennenzulernen (vgl. mit Forderung nach *Ideenanregung*, [Romeike, 2007b]), da so davon auszugehen ist, dass Absolventen zum einen diese wichtige Fähigkeit weiter ausbauen können, zum anderen aber auch aktiv davon berichten und so ein positiveres Ansehen unter jüngeren Personengruppen verbreiten. Beispielsweise konnte gezeigt werden, dass ein projektbegleitender Kreativitätsraum, der von Studierenden mitgestaltet wird, in der Lage ist, soziale und kreative Aspekte in den Ergebnissen zu veranschaulichen [Göttel und Schild, 2011] und so z.B. Bekannte und Verwandte der Projektteilnehmer von diesen attraktiven Eigenschaften der Informatik zu überzeugen. Aber auch Strukturänderungen und Spezialisierungen von Informatik-Studiengängen, wie sie z.B. am Fachbereich Informatik der Universität Hamburg zu sehen sind[5], können dazu beitragen ein Bild der Informatik entstehen zu lassen, das der Realität entsprechend viele verschiedene

[5]Der Fachbereich Informatik der Universität bietet mit großem Erfolg bzw. Zulauf seit dem Wintersemester 2009/2010 neben Informatik und Wirtschaftsinformatik (BSc.) noch weitere spezialisierte BSc.-Studiengänge an (Mensch-Computer-Interaktion, Software-System-Entwicklung und Computing in Science).

Aspekte besitzt und für verschiedenste Interessen bzw. Fähigkeiten Möglichkeiten und Zukunftsperspektiven bietet.

Die Interdisziplinarität und die Praxisorientiertheit sind weitere Aspekte, die zum Selbstbild der Informatik gehören (siehe Kapitel 2.1.1). Dementsprechend sollte auch geprüft werden, auf welche Art diese Faktoren Jugendlichen zugänglich gemacht werden können. Denkbar wären Mentorenprogramme, in denen von realen und interdisziplinären Projekten berichtet wird oder gar Arbeiten in solchen Projekten gemeinsam mit Schülern durchgeführt werden können. Aus der Games Industrie kommen dazu interessante erste Ansätze, wie beispielsweise „Learn With Portals“[6], bei dem Schulklassen Entwicklerstudios besuchen und dort im Kontext eines realen kommerziellen Spiels Aufgaben erhalten und gemeinsam mit Mitarbeitern aus verschiedenen Fachrichtungen zusammenarbeiten und von diesen betreut werden. Jugendliche haben so in einem für sie attraktiven Kontext die Chance, reale Bedingungen und Aufgaben kennenzulernen, und können damit einen Bezug zur Informatik herstellen.

Betrachtet man die aktuellen Trends der Unterhaltungselektronik und beispielsweise auch der *Child Computer Interaction* hin zur Gesten- und Körpersteuerung oder dem *full body engagement* (vergleiche beispielsweise Soler-Adillon und Parés [2009]), so scheinen auch hier noch viele ungeahnte Wege zu existieren, die Jugendlichen informatische Probleme wirklich greifbar und attraktiv vermitteln könnten (z.B. Merrill et al. [2007]). Beispielsweise verzichtet im Gegenzug Gallenbacher [2008] bewusst vollständig auf Technologie und vermittelt Informatik statt dessen mit Bastelbögen[7]. Auch hier verbirgt sich sicherlich eine spannende Möglichkeit, in einem weiterführenden Schritt Bastelbögen technisch zu erweitern und aufzuwerten, wobei das Greifbare und Erfahrbare erhalten bliebe[8].

Neben Ausprägungen der Softwareentwicklung gibt es auch das weite Feld der Robotik/Roboterprogrammierung als Attraktor für Informatik-Studiengänge. So eignen sich beispielsweise LEGO Mindstorms und PicoCrickets mit ihren zugehörigen grafischen Programmierumgebungen gut, um abermals Kreativität in der Informatik hervorzuheben und jugendliche Interessen gezielt anzusprechen (vergleiche beispielsweise [Romeike und Reichert, 2011]), um sie dann während des Studiums weiter zu fördern.

[6] http://www.learnwithportals.com/, zuletzt besucht am 15. April 2012.

[7] Dies ist eine Ausprägung des aktuell im Bildungsbereichs viel beachteten Konzepts *Computer Science Unplugged*: http://csunplugged.org/, zuletzt besucht am 15. April 2012.

[8] Erste Vorschläge bzw. Umsetzungen sind beispielsweise bei *Greenfoot* zu finden: http://www.greenfoot.org/doc/csunplugged, zuletzt besucht am 15. April 2012.

Im allgemeinen Unterricht, also nicht zwangsläufig im Informatikunterricht, sollten in Projekten und speziellen Angeboten verstärkt kulturelle Auswirkungen der IT thematisiert und erfahrbar gemacht werden. Damit bestünde die Gelegenheit, zum einen auf Gefahren hinzuweisen, jedoch zum anderen auch das positive Bild nach Rolf [2008a] (vergleiche Kapitel 2.3) zu vermitteln, dass die IT eine große Chance darstellt, aktiv zukünftige Kultur und Gesellschaft mitzugestalten. Somit würde sich auch die Möglichkeit ergeben, negativ wahrgenommene Ökonomisierungen nicht immer zwangsläufig und ausschließlich der IT zuzuschreiben, wie es ebenfalls Rolf vermutet.

Nicht zuletzt sollte die IT-Branche sich immer wieder bezüglich ihrer Außenwirkung hinterfragen und ihre Systeme und ihren Service den Praktikern intuitiv zugänglich machen (vergleiche Denning [2002], Kapitel 2.3). Es handelt sich hierbei jedoch um einen langwierigen Prozess, der nicht zentral gesteuert werden kann und daher nur schwer aus akademischer Sicht zu beeinflussen ist. Der Fakt, dass agile Methoden immer mehr auch in größeren Unternehmen eingesetzt werden, gibt Anlass zur Hoffnung, dass hier ein Umdenken eingesetzt hat.

## 10.3 Abschließende Worte

Die Softwareentwicklung stellt nur einen von vielen Teilbereichen in der Informatik dar. Idealerweise sollten zukünftig weitere Maßnahmen für andere Unterdisziplinen der Informatik identifiziert und umgesetzt werden, um die IT insgesamt in der gesellschaftlichen Wahrnehmung attraktiver erscheinen zu lassen. Auch von der Gruppe der Jugendlichen abgesehen, existieren viele verschiedene potenzielle Nutzerdomänen, denen dedizierte Zugänge zur IT vermittelt werden sollen.

Nichts desto trotz bietet die vorliegende Arbeit eine Grundlage, um den Herausforderungen zu begegnen und die Attraktivität der Informatik maßgeblich zu steigern. Die intensive Bearbeitung der Thematik und die umfassende Auseinandersetzung mit Schülern in der Praxis haben zur Erarbeitung eines integrierten Konzepts geführt, das es erlaubt, durch Erweiterungen an *initial learning environments* und methodisch-didaktische Vorgehensweisen deutlicher auf die sozialen Aspekte der Softwareentwicklung hinzuweisen. So kann Jugendlichen früh ein reales und attraktives Bild der Softwareentwicklung aufgezeigt werden. Der virtuelle Sandkasten unterstützt technisch soziale Gruppenprozesse, wohingegen die angepassten agilen Methoden die Gruppenarbeit strukturieren und einen Bezug zur modernen Softwareentwicklung herstellen. In der Kombination erlauben diese beiden Maßnahmen

aktiv ein positives Image der Informatik unter Jugendlichen im Schulkontext zu fördern. Deren Einbindung in den Informatikunterricht stellt einen vielversprechenden, für Jugendliche und deren soziale Entwicklung geeigneten Ansatz dar, um dem Nachwuchsproblem der Informatik erfolgreich entgegen zu wirken.

# Anhang A

# Pair programming in SE1, Fachbereich Informatik, Universität Hamburg, WiSe 08/09

## A.1 Fragebogen

Dieser Bogen wird maschinell ausgewertet. Bitte korrigiert fehlerhafte Anworten ungefähr so: ○●✕ .

# Umfrage zum Programmieren im Paar in den SE1-Übungen

Wir verwenden in SE1 das Programmieren im Paar (PP) zu Lehrzwecken, es handelt sich dabei um eine Methode aus dem Extreme Programming, einem Vorgehensmodell in der professionellen Software-Entwicklung (SE). Dieser Fragebogen soll gezielt auf den Aspekt von PP in der Lehre eingehen.

## Persönliche Daten

| | |
|---|---|
| Selbstgewählter anonymer Name: | |
| Studiengang: | ○ BSc Informatik ○ BSc Wirtschaftsinf. ○ Dipl. Informatik ○ Dipl. Wirtschaftsinf. ○ Lehramt ○ Sonstige ☐ |
| Semester | ○ 1 ○ 3 ○ 5 ○ >=7 |
| Alter | ○ < 21 ○ 21 - 25 ○ 26 - 30 ○ > 30 |
| Geschlecht | ○ weiblich ○ männlich |

## Programmieren im Paar in den SE1-Übungen

| | |
|---|---|
| Meine Programmiererfahrungen vor dieser Veranstaltung erwarb ich… | ○ in der Schule. ○ durch einen Job. ○ durch Selbststudium. ○ gar nicht; ich hatte keine. |
| War Dir die Idee von PP vor SE1 bekannt? | ○ ja ○ nein |
| Bevorzugst Du es, die Aufgaben alleine zu lösen, wenn es möglich ist? | ○ ja ○ nein |
| Fördert PP Dein Verständnis für den Stoff von SE1? | ○ ja, es hilft mir sehr. ○ neutral, es ist schwer zu beurteilen. ○ nein, ich halte es nicht für hilfreich. |
| Entsteht durch PP ein Gemeinschaftsgefühl für das Ergebnis? | ○ ja und das ist ein Mehrwert. ○ ja aber das bringt mir nichts. ○ nein. |
| Fördert Deiner Meinung nach PP das allgemeine Gemeinschaftsgefühl unter den Teilnehmern? | ○ ja ○ nein |
| Hast Du das Gefühl, die Aufgaben wirklich im Paar gelöst zu haben? | ○ ja, beide Partner trugen im Schnitt in gleichen Maßen zum Ergebnis bei. ○ nein, ich habe im Schnitt weniger getan als mein Partner. ○ nein, ich habe im Schnitt mehr getan als mein Partner. |
| Bist Du der Meinung, dass das PP aus unserer Veranstaltung geeignet ist, um Werbung für den Studiengang Informatik zu machen? | ○ ja, die soziale Komponente von PP macht Informatik definitiv attraktiver. ○ PP ist zwar hilfreich, aber kein wichtiger Aspekt des Studiengangs. ○ nein, PP hat keinen Einfluss auf die Attraktivität des Studiengangs. ○ nein, PP stößt mich eher ab, auf keinen Fall damit Werbung machen. |

Zeitreise: Was war Dein Eindruck von PP nach dem ersten Übungstermin? Hat sich seit dem etwas geändert? Wenn ja, was?

Nur für Personen mit Programmiervorerfahrungen: Hat Dir PP Lust darauf gemacht, solche Methoden auch weiter in anderen SE-Projekten einzusetzen (bitte mit kurzer Begründung)?

## A.2 Ergebnisse

# Auswertung zur Veranstaltung "SE 1"

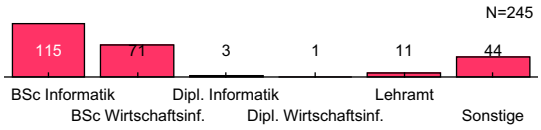
Liebe Dozentin, lieber Dozent,
anbei erhalten Sie die Ergebnisse der Evaluation Ihrer Lehrveranstaltung.
Zu dieser Veranstaltung wurden 254 Bewertungen (bei 300 TeilnehmerInnen) abgegeben. Dies entspricht einer Rücklaufquote von 85%.
Erläuterungen zu den Diagrammen befinden sich am Ende dieses Dokuments. Die Auswertungen aller Veranstaltungen, die von mehr als fünf TeilnehmerInnen evaluiert wurden, sind — ohne persönliche Kommentare — in einigen Tagen unter der URL

https://www.blubbsoft.de/evaluation

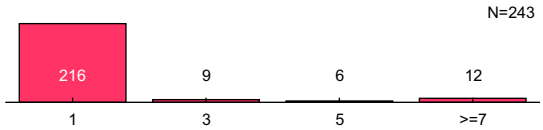
verfügbar. Mit freundlichen Grüßen,
Das Evaluationsteam

## Umfrage zum Programmieren im Paar in den SE1-Übungen

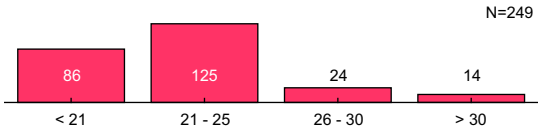
### Persönliche Daten



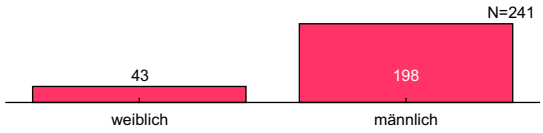

Studiengang:




Semester

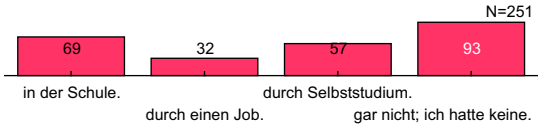



Alter

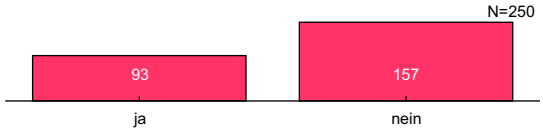



Geschlecht

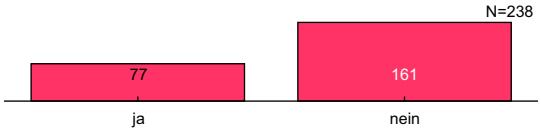
### Programmieren im Paar in den SE1-Übungen



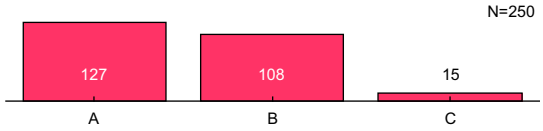

Meine Programmiererfahrungen vor dieser Veranstaltung erwarb ich…



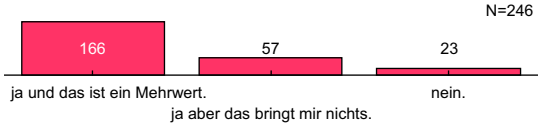

War Dir die Idee von PP vor SE1 bekannt?



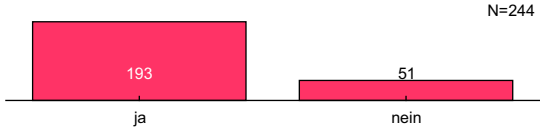

Bevorzugst Du es, die Aufgaben alleine zu lösen, wenn es möglich ist?



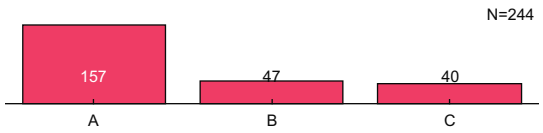

Fördert PP Dein Verständnis für den Stoff von SE1?
**A** ja, es hilft mir sehr.
**B** neutral, es ist schwer zu beurteilen.
**C** nein, ich halte es nicht für hilfreich.




Entsteht durch PP ein Gemeinschaftsgefühl für das Ergebnis?

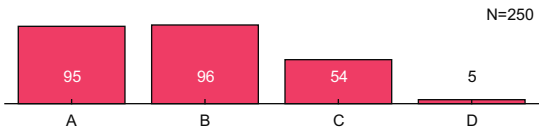



Fördert Deiner Meinung nach PP das allgemeine Gemeinschaftsgefühl unter den Teilnehmern?




Hast Du das Gefühl, die Aufgaben wirklich im Paar gelöst zu haben?

**A** ja, beide Partner trugen im Schnitt in gleichen Maßen zum Ergebnis bei.
**B** nein, ich habe im Schnitt weniger getan als mein Partner.
**C** nein, ich habe im Schnitt mehr getan als mein Partner.




Bist Du der Meinung, dass das PP aus unserer Veranstaltung geeignet ist, um Werbung für den Studiengang Informatik zu machen?

**A** ja, die soziale Komponente von PP macht Informatik definitiv attraktiver.
**B** PP ist zwar hilfreich, aber kein wichtiger Aspekt des Studiengangs.
**C** nein, PP hat keinen Einfluss auf die Attraktivität des Studiengangs.
**D** nein, PP stößt mich eher ab, auf keinen Fall damit Werbung machen.

# Anhang B

# Fragebogen, GI HILL, 2008

Die Fragen dienen dazu, herauszufinden welche computergestützten Hilfen im praktischen Einsatz für Schüler und Lehrer von Nutzen wären. Da Informatik in verschiedenen Kontexten in Schulen angeboten wird, ist es **möglich** bei der **zweiten Frage mehrere Felder** anzukreuzen. Für die darauf folgenden Fragen ist es möglich, den **kennzeichnenden Buchstaben** aus Frage 2 für den die Antwort zutrifft, einzutragen (somit sind auch dort Mehrfachantworten möglich). Auch bei den offenen Antworten wäre es für die Auswertung hilfreich, die Buchstaben in Klammern mit anzugeben.

1. ***Schulform***
   - ( ) Gymnasium
   - ( ) Gesamtschule
   - ( ) andere Form: _____

2. ***Informatikinhalte werden dort von mir angeboten als***
   - ( A ) Pflichtfach, Std/Woche: _____
   - ( B ) Wahlpflicht, Std/Woche: _____
   - ( C ) Wahlfach, Std/Woche: _____
   - ( D ) AG, Std/Woche: _____
   - ( E ) Projektwocheninhalt

3. ***Ausstattung***
   - ( ) je Schüler ist ein PC vorhanden
   - ( ) auf ca. zwei Schüler kommt ein PC
   - ( ) auf ca. vier Schüler kommt ein PC
   - ( ) auf über vier Schüler kommt ein PC

4. ***Welche Inhalte werden vorwiegend programmiert?***
   - ( ) Anwendungen (z.B. DVD-Verwaltung)
   - ( ) Algorithmen
   - ( ) Computerspiele
   - ( ) Anderes: _____

5. ***Verwendete Programmiersprachen***
   - ( ) Java
   - ( ) C / C++
   - ( ) Squeak
   - ( ) Scripting (Python, PHP, Ruby, o.ä.)
   - ( ) Pascal
   - ( ) Andere: _____

6. ***Benutzte Entwicklungsumgebungen***
   - ( ) BlueJ
   - ( ) Greenfoot
   - ( ) Eclipse / NetBeans
   - ( ) Text-Editor: _____
   - ( ) Wahl bleibt den Schülern überlassen

7. ***Welche Web-Plattformen zur Unterstützung der Projektarbeit werden verwendet?***
   - ( ) (Schul-)CommSy
   - ( ) BSCW
   - ( ) lo-net$^2$
   - ( ) Fronter
   - ( ) Andere: _____
   - ( ) Keine

8. ***In welcher Form gibt es Gruppenarbeit im Informatikunterricht (Gibt es beispielsweise Aufgaben, deren Teilaufgaben innerhalb eines Programmierteams verteilt werden, wie groß sind solche Teams, usw.)?***

_____

_____

_____

_____

_____

9. ***Ist es üblich, gegen Ende einer Schulstunde den Schülern Zeit zur freien Verfügung an den PCs anzubieten? Was wird dabei von den Schülern genutzt (z.B. Programme, Spiele, Internetanwendungen)?***

( ) ja ( ) nein

_____

_____

Bitte wenden…

***10. In welchen Situationen fehlt es an computergestützter Hilfe oder Koordination für bzw. von Programmierteams?***

***11. Wie sollte so eine Unterstützung aussehen?***

***12. Ist es den Schülern erlaubt, Daten für die Programmieraufgaben untereinander auszutauschen? Auf welchem Wege geschieht dies (z.B. E-Mail, USB-Stick, Online-Plattform, usw.) und wird es geregelt / beaufsichtigt?***

( ) ja ( ) nein

***13. Wird von den Schülern verlangt ihre Programme zu dokumentieren? Wie ist der „normale“ Ablauf der Ausarbeitung der Dokumentation? Wie ist die Einstellung der Schüler gegenüber der Anfertigung von Dokumentationen?***

( ) ja ( ) nein

***14. Wenn bei Frage 7 mind. eine Online-Plattform angegeben wurde: Wie ist das Online-Verhalten der Schüler zu beschreiben? Wie hoch ist die Aktivität während einem Projekt und wie hoch danach? Sind Moderation bzw. Anregungen durch Lehrer nötig?***

Persönliches, natürlich anonym:

( ) männlich ( ) weiblich Alter ( )

Informatiklehrer ( ) andere Fachrichtung: _____

Unterricht in Informatik seit: _____

Informatik / Programmieren in der Freizeit: ja ( ) nein ( )

Kontakt:
E-Mail: goettel@informatik.uni-hamburg.de
Tel.: 42883 2371

# Anhang C

# Scratch Aufgabenblatt

# SCRATCH KENNENLERNEN

Da kommt leider ziemlich viel Text auf Dich zu! **Lass Dir Zeit**. Du solltest Dir ruhig häufiger eine **Auszeit** von den Aufgaben gönnen und einfach nur so ein mit den Bausteinen von Scratch experimentieren oder auch andere Projekte auf www.scratch.mit.edu ausprobieren, es soll ja schließlich Spaß machen. Manche Aufgaben schlagen Dir noch Scratch-Karten vor, die man mal angucken könnte, um ähnliche Verhaltensweisen in Scratch zu erstellen.

## AUFGABE 1: „HEKTOR UND ANNA ERSCHEINEN AUF DER BÜHNE“

Füge ein neues Objekt hinzu und gebe ihm den Namen „Hektor“. Danach änderst Du das Kostüm der Katze und gibst diesem Objekt den Namen „Anna“. Da das Katzenkostüm jetzt nicht mehr benötigt wird, kannst Du es löschen. Achte bitte darauf, dass beide Figuren ungefähr gleich groß sind und passe sie gegebenenfalls aneinander an.

Sobald man das Programm durch Klicken auf die grüne Fahne beginnt, sollen sich Anna und Hektor an bestimmten Orten auf der Bühne befinden. Und zwar Anna an der Position x: -180 und y:-20, Hektor dagegen an der Stelle x: 180 und y: -10.

Zum Schluss der Aufgabe kannst Du Dir noch einen passenden Hintergrund aussuchen. Dies ist so ähnlich wie ein Kostüm zu wechseln...

*Benötigte Bausteine (u.a.):*


*Scratch-Karte zum Experimentieren:*

Nr. 1 „Farbe ändern“

## AUFGABE 2: „HEKTOR LERNT LAUFEN“

Lasse Hektor 100 Schritte auf Anna zugehen (auch hier wieder nur, wenn das Programm durch einen Klick auf die Fahne gestartet wurde). Achtung, die Aufgabe klingt leicht, es gibt dabei jedoch einige Dinge zu beachten:

- Du musst Dich zuerst darauf festlegen, wie sich Hektor drehen soll, momentan dreht er sich nämlich im Kreis. Da wir ihn aber nur von rechts nach links oder umgekehrt laufen lassen wollen, musst Du eine Änderung vornehmen. Wenn Du Hektor ausgewählt hast siehst Du wie hier abgebildet eine kurze Info, dort kannst Du aber auch auswählen wie er sich drehen soll. Momentan ist der obere Knopf hellblau markiert. Klicke nun einfach auf den mittleren Knopf, so dass dieser hellblau erscheint, wie auf dem Bild unten zu erkennen. Damit hast Du festgelegt, dass er sich nur von rechts nach links (und umgekehrt) umdrehen wird (Tipp: Es ist zu Anfang immer leichter, wenn man diese Einstellung für alle Objekte vornimmt).

- Du solltest Hektor genau sagen in welche Richtung er zeigt damit er auch in die richtige Richtung losgehen kann, sobald Du ihm das „befiehlst“. Mit (-90) zeigt er nach links und wird sich dann auch in diese Richtung bewegen. (+90) lässt ihn nach rechts zeigen. Übrigens kannst Du seine momentane Ausrichtung auch immer bei den im oberen Bild gezeigten Informationen ablesen.

- So, nun kommen wir endlich zur eigentlichen Aufgabe:

  Es gibt zwei Möglichkeiten, um 100 Schritte vorwärts zu gehen, entweder entscheidest Du dich für den Baustein „gehe 10 -er Schritt“ und ersetzt die 10 durch eine 100. Das ist aber nicht so schön, da dann Hektor sofort um 100 Schritte in die gewählte Richtung springt. Eleganter wäre es doch wenn man ihm zuschauen kann, wie er auf Anna zugeht. Das erreichst Du, indem man in mehrmals nacheinander kleine Schritte gehen lässt, z.B. hundertmal  Einer-Schritte. Hast Du eine Idee, wie man das machen könnte? Die gezeigten Bausteine sollten Dir weiterhelfen…denke daran, Hektor soll loslaufen, sobald jemand auf die grüne Fahne geklickt hat.

*Benötigte Bausteine (u.a.):*



*Scratch-Karten zum Experimentieren:*

Nr. 2 „Bewege Dich zum Rhythmus“
Nr. 3 „Tastenbewegung“
Nr. 5 „Gleiten“
Nr. 6 „Folge der Maus“
Nr. 10 „Animierte Bewegung“

## AUFGABE 3: „HEKTOR STELLT DEN ERSTEN KONTAKT HER“

Er sagt zu ihr für zwei Sekunden „Hallöchen!“ und sendet eine entsprechende Nachricht, auf die Anna wie folgt reagiert: Zuerst denkt sie für zwei Sekunden „Ihhh!“ und dann sagt sie für 3 Sekunden „…äh, was willst Du?“. Auch sie sendet eine entsprechende Nachricht, auf die Hektor reagieren kann.

*Benötigte Bausteine (u.a.):*



So eigentlich hast Du jetzt erst einmal **genug gelernt**, um eine **eigene Geschichte** mit Scratch zu erzählen. Die Aufgaben drei und vier sind etwas schwieriger und können ruhig warten.

...

## Aufgabe 3: „Hektor will gefunden werden“

Lass Hektor auf Annas „äh“ wie folgt reagieren: Er gibt damit an, dass er sich unsichtbar machen kann und sagt „Guck mal was ich kann! Kannst Du mich finden?“. Daraufhin versteckt er sich und geht an einen zufälligen Ort entlang der x-Position. Hierbei musst Du darauf achten, dass er beim nächsten Neustart durch Klicken auf die Fahne wieder sichtbar wird.

*Benötigte Bausteine (u.a.):*


*Scratch-Karte zum Experimentieren:*

Nr. 11 „Überraschungsschalter“

## Aufgabe 4: „Die Suche nach Hektor“

Erstelle eine Variable mit dem Namen hektor_sichtbar, die 0 ist wenn er sich versteckt und 1 ist wenn er wieder auftaucht bzw. wenn er sichtbar ist. Sollte Anna angeklickt werden und Hektor nicht sichtbar sein, so fragt sie „Wo bist du bloß?“ ansonsten denkt sie beim anklicken „Das ist aber ein komischer Typ…“.

Damit Hektor durch Klicken auf ihn (im versteckten Zustand) wieder auftaucht, musst Du wie folgt vorgehen: Nachdem Hektor nicht mehr sichtbar ist, muss man immer wieder darauf warten, dass die Maustaste gedrückt wird. In diesem Fall muss man überprüfen, ob sich der Mauszeiger zum Zeitpunkt des Klickens über dem versteckten Hektor befunden hat. Zum Beispiel sollte man dafür diese Werte benutzen: x-30 < Maus-x-Position und Maus-x-Position < x+30. Sollte sich der Mausklick innerhalb dieses Wertes ereignet haben, so kannst Du Hektor wieder auftauchen lassen und ihn noch etwas lobend dazu sagen lassen.

*Benötigte Bausteine (u.a.):*

# Literaturverzeichnis


Adlin, T. und Pruitt, J. (2010). *The Essential Persona Lifecycle: Your Guide to Building and Using Personas*. Morgan Kaufmann.

Al-Bow, M., Austin, D., Edgington, J., Fajardo, R., Fishburn, J., Lara, C., Leutenegger, S., und Meyer, S. (2008). Using Greenfoot and Games to Teach Rising 9th and 10th Grade Novice Programmers. In *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games*, Sandbox '08, S. 55–59. ACM. DOI: 10.1145/1401843.1401853.

Anger, C., Erdmann, V., und Plünnecke, A. (2011). MINT - Trendreport 2011. Report, Institut der deutschen Wirtschaft Köln.

Antil, L. R., Jenkins, J. R., und Wayne, S. K. (1998). Cooperative Learning: Prevalence, Conceptualizations, and the Relation Between Research and Practice. *American Educational Research Journal*, 35(3):419–454. DOI: 10.3102/00028312035003419.

Antle, A. N. (2006). Child-Personas: Fact or Fiction? In *Proceedings of the 6th Conference on Designing Interactive Systems*, DIS '06, S. 22–30. ACM. DOI: 10.1145/1142405.1142411.

Antle, A. N. (2008). Child-Based Personas: Need, Ability and Experience. *Cognition, Technology & Work*, 10(2):155–166. DOI: 10.1007/s10111-007-0071-2.

Ashworth, F., Brennan, G., Egan, K., Hamilton, R., und Sáenz, O. (2004). Learning Theories and Higher Education. *Level 3*, (2).

Bailey, J. L. und Stefaniak, G. (2001). Industry Perceptions of the Knowledge, Skills, and Abilities Needed by Computer Programmers. In *Proceedings of the 2001 ACM SIGCPR Conference on Computer Personnel Research*, SIGCPR '01, S. 93–99. ACM. DOI: 10.1145/371209.371221.

Baumann, R. (1996). *Didaktik der Informatik*. Klett.

Baumann, R. (2009a). Propädeutische Algorithmik und Objektorientierung mit Etoys. *LOG IN*, (160/161).

Baumann, R. (2009b). Sprechende Katze und Zeichenschildkröte. *LOG IN*, (156).

Baumgartner, P. und Himpsl, K. (2008). Auf dem Weg zu einer neuen Lernkultur. *LOG IN*, (152).

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman.

Beck, K. und Andres, C. (2004). *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Longman.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., und Thomas, D. (2001). Agile Manifesto. `http://agilemanifesto.org/`.

Beedle, M., Devos, M., Sharon, Y., Schwaber, K., und Sutherland, J. (1999). Scrum: A Pattern Language for Hyperproductive Software Development. In Harrison, N., Foote, B., und Rohnert, H., Hrsg., *Pattern Languages of Program Design 4*, S. 637–651. Addison-Wesley Longman.

Benamati, J. S. (2007). Current and Future Entry-Level IT Workforce Needs in Organizations. In *Proceedings of the 2007 ACM SIGMIS CPR Conference on Computer Personnel Research: The Global Information Technology Workforce*, SIGMIS CPR '07, S. 101–104. ACM. DOI: 10.1145/1235000.1235024.

Berenson, S. B., Slaten, K. M., Williams, L., und Ho, C.-W. (2004). Voices of Women in a Software Engineering Course: Reflections on Collaboration. *Journal on Educational Resources in Computing*, 4(1). DOI: 10.1145/1060071.1060074.

Bergstrom, T. und Karahalios, K. (2009). Vote and Be Heard: Adding Back-Channel Signals to Social Mirrors. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part I*, INTERACT '09, S. 546–559. Springer. DOI: 10.1007/978-3-642-03655-2_61.

Borriello, G. (2006). The Inivisble Assistant. *Queue*, 4(6):44–49. DOI: 10.1145/1147518.1147532.

Braught, G., Wahls, T., und Eby, L. M. (2011). The Case for Pair Programming in the Computer Science Classroom. *Transactions on Computing Education*, 11(1):2:1–2:21. DOI: 1921607.1921609.

Bruckman, A. (1998). Community Support for Constructionist Learning. *Computer Supported Cooperative Work*, 7(1-2):47–86. DOI: 10.1023/A:1008684120893.

Burnett, M., Fleming, S. D., Iqbal, S., Venolia, G., Rajaram, V., Farooq, U., Grigoreanu, V., und Czerwinski, M. (2010). Gender Differences and Programming Environments: Across Programming Populations. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. DOI: 10.1145/1852786.1852824.

Carter, L. (2011). Ideas for Adding Soft Skills Education to Service Learning and Capstone Courses for Computer Science Students. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, S. 517–522. ACM. DOI: 10.1145/1953163.1953312.

Caspersen, M. E. und Kölling, M. (2009). STREAM: A First Programming Process. *Transactions on Computing Education*, 9(1):4:1–4:29. DOI: 10.1145/1513593.1513597.

Clayton, K. L., von Hellens, L. A., und Nielsen, S. H. (2009). Gender Stereotypes Prevail in ICT: a Research Review. In *Proceedings of the Special Interest Group on Management Information System's 47th Annual Conference on Computer Personnel Research*, SIGMIS CPR '09, S. 153–158. ACM. DOI: 10.1145/1542130.1542160.

Cooper, A. (1999). *The Inmates Are Running the Asylum*. Macmillan Publishing.

Coy, W. (2001). Was ist Informatik? In Desel, J., Hrsg., *Das ist Informatik*, Kapitel 1, S. 1 – 22. Springer.

Degele, N. (2002). *Einführung in die Techniksoziologie*. Wilhelm Fink Verlag.

Denning, P. J. (2002). When IT becomes a profession. In Denning, P. J., Hrsg., *The invisible future*, Kapitel 23, S. 295–325. McGraw-Hill.

Döbeli Honegger, B. (2007). Wiki und die starken Potenziale. *Computer + Unterricht*, 17(66):39–41.

Druin, A. (2002). The Role of Children in the Design of New Technology. *Behaviour & Information Technology*, 21(1):1–25. DOI: 10.1080/01449290110108659.

Druin, A. und Fast, C. (2002). The Child as Learner, Critic, Inventor, and Technology Design Partner: An Analysis of Three Years of Swedish Student Journals. *International Journal of Technology and Design Education*, 12(3):189–213. DOI: 10.1023/A:1020255806645.

Druin, A., Smith, D., Huchital, J., Chanover, M., und Bruckman, A. (1997). Computers, Kids, and Creativity: What Does the Future Hold? In *CHI 97 Extended Abstracts on Human Factors in Computing Systems*, S. 111–112. ACM. DOI: 10.1145/1120212.1120284.

Engbring, D. (2011). Untersuchungen und Bewertungen zum Einsatz von Alice im Informatikunterricht. In *Informatik mit Kopf, Herz und Hand. Praxisbeiträge zur INFOS 2011*, S. 81–90. ZfL-Verlag.

Fachbereich Informatik der Universität Hamburg (2011). Jahresbericht 2010.

Fincher, S. und Utting, I. (2010). Machines for Thinking. *Transactions on Computing Education*, 10(4):13:1–13:7. DOI: 10.1145/1868358.1868360.

Finck, M., Janneck, M., Janneck, M., und Obendorf, H. (2005). Kooperative Wissensnetze. In *Mensch & Computer 2005: Kunst und Wissenschaft - Grenzüberschreitungen der interaktiven ART*, S. 133–142.

Fisker, K., McCall, D., Kölling, M., und Quig, B. (2008). Group Work Support for the BlueJ IDE. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '08, S. 163–168. ACM. DOI: 10.1145/1384271.1384316.

Floyd, C., Kelkar, G., Klein-Franke, S., Kramarae, C., und Limpangog, C., Hrsg. (2002). *Feminist Challenges in the Information Age*. Leske + Budrich.

Fothe, M. und Friedrich, S. (2011). Informatik in die Schule! - ein erneutes Plädoyer. `http://www.gi.de/fileadmin/redaktion/Vorstandsglossen/GI-Vorstandsmitglied-Fothe110523.pdf`.

Fuchs, C. und Hofkirchner, W. (2003). *Studienbuch Informatik und Gesellschaft*. Libri Books on Demand.

Gallenbacher, J. (2008). *Abenteuer Informatik. IT zum Anfassen - von Routenplaner bis Online-Banking*. Spektrum Akademischer Verlag.

García-Crespo, A., Palacios, R. C., Berbís, J. M. G., und Caro, E. T. (2009). IT Professionals' Competences: High School Students' Views. *Journal of Information Technology Education*, 8:45–57.

Garzotto, F. (2008). Broadening Children's Involvement as Design Partners: From Technology to Experience. In *Proceedings of the 7th International Conference on Interaction Design and Children*, S. 186–193. ACM. DOI: 10.1145/1463689.1463755.

Gesellschaft für Informatik e.V. (2000). *Empfehlungen für ein Gesamtkonzept zur informatischen Bildung an allgemein bildenden Schulen*. Gesellschaft für Informatik e.V. (GI).

Gesellschaft für Informatik e.V. (2006). *Was ist Informatik? Unser Positionspapier*. Gesellschaft für Informatik e.V. (GI).

Gesellschaft für Informatik e.V. (2008). *Grundsätze und Standards für die Informatik in der Schule*. Gesellschaft für Informatik e.V. (GI).

Göttel, T. (2009a). Culturia: Intercultural Learning by Designing Games. In Turgeon, W. C., Hrsg., *Creativity and the Child: Interdisciplinary Perspectives*. The Inter-Disciplinary Press.

Göttel, T. (2009b). Virtual Sandbox: Adding Groupware Abilities to Scratch. In *Proceedings of the 8th International Conference on Interaction Design and Children*, S. 158–161. ACM. DOI: 10.1145/1551788.1551816.

Göttel, T. (2011a). Agiler Informatikunterricht: Soziale Aspekte der professionellen Softwareentwicklung im Schulunterricht erfolgreich erfahrbar machen. In *Informatik in Bildung und Beruf. 14. GI-Fachtagung Informatik und Schule - INFOS 2011*, S. 37–46. Gesellschaft für Informatik e.V. (GI).

Göttel, T. (2011b). Child-Personas and Agile Methods in Class to Focus on Social Aspects of Computer Science. In Workshop *Opportunities and Challenges when Designing and Developing with Kids @ School*. `http://workshops.icts.sbg.ac.at/KidsSchool/papers/7_Goettel_IDC11.pdf`.

Göttel, T. (2011c). Reviewing Children's Collaboration Practices in Storytelling Environments. In *Proceedings of the 10th International Conference on Interaction Design and Children*, S. 153–156. ACM. DOI: 10.1145/1999030.1999049.

Göttel, T. und Schild, J. (2011). Creativity Room 5555: Evoking Creativity in Game Design amongst CS Students. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, S. 98–102. ACM. DOI: 10.1145/1999747.1999777.

Gutwin, C., Greenberg, S., und Roseman, M. (1996). Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation. In *Proceedings of HCI on People and Computers XI*, S. 281–298. Springer.

Hall, T., Wilson, D., Rainer, A., und Jagielska, D. (2007). Communication: the Neglected Technical Skill? In *Proceedings of the 2007 ACM SIGMIS CPR Conference on Computer Personnel Research: The Global Information Technology Workforce*, SIGMIS CPR '07, S. 196–202. ACM. DOI: 10.1145/1235000.1235043.

Harel, I. (1991). *Children Designers: Interdisciplinary Constructions for Learning and Knowing Mathematics in a Computer-Rich School*. Ablex Publishing Corporation.

Hazzan, O. und Dubinsky, Y. (2007). Why Software Engineering Programs Should Teach Agile Software Development. *SIGSOFT Software Engineering Notes*, 32(2):1–3. DOI: 10.1145/1234741.1234758.

Hellige, H.-D. (2008). Die Geschichte des Internet als Lernprozess. In Kreowski, H.-J., Hrsg., *Informatik und Gesellschaft, Verflechtungen und Perspektiven*, S. 121–170. Lit Verlag.

Henriksen, P., Kölling, M., und McCall, D. (2010). Motivating Programmers via an Online Community. *Journal of Computing Sciences in Colleges*, 25(3):82–93.

Homberg, G. (2006). Selbstgesteuertes Lernen als kooperativer Prozess. *LOG IN*, (138/139).

Hubwieser, P. (2007). *Didaktik der Informatik: Grundlagen, Konzepte, Beispiele*. Springer.

Humbert, L. (1999). Kollaboratives Lernen. *LOG IN*, (3/4).

Jackewitz, I., Janneck, M., und Strauss, M. (2004). CommSy: Softwareunterstützung für Wissensprojekte. In Pape, B., Krause, D., und Oberquelle, H., Hrsg., *Wissensprojekte - Gemeinschaftliches Lernen aus didaktischer, softwaretechnischer und organisatorischer Sicht*, S. 186–202. Waxmann.

Janneck, M. (2004). Projektorientierung. In Haake, J., Schwabe, G., und Wessner, M., Hrsg., *CSCL-Kompendium*, S. 238–244. Oldenbourg Verlag.

Janneck, M. (2006). Partizipative Systementwicklung im Informatikunterricht. *LOG IN*, (138/139).

Janneck, M. und Janneck, M. (2004). Gruppen und Gruppenarbeit. In Haake, J., Schwabe, G., und Wessner, M., Hrsg., *CSCL-Kompendium*, S. 42–53. Oldenbourg Verlag.

Johansen, R., Sibbet, D., Benson, S., Martin, A., Mittman, R., und Saffo, P. (1991). *Leading Business Teams: How Teams Can Use Technology and Group Process Tools to Enhance Performance*. Addison-Wesley Longman.

Kabicher, S., Motschnig-Pitrik, R., und Figl, K. (2009). What Competences do Employers, Staff and Students Expect from a Computer Science Graduate? In *Frontiers in Education Conference, 2009. FIE '09.*, S. 1–6. IEEE. DOI: 10.1109/FIE.2009.5350536.

Keefe, K., Sheard, J., und Dick, M. (2006). Adopting XP Practices for Teaching Object Oriented Programming. In *Proceedings of the 8th Austalian conference on Computing education - Volume 52*, ACE '06, S. 91–100. Australian Computer Society.

Kelleher, C. und Pausch, R. (2005). Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Computing Surveys*, 37(2):83–137. DOI: 10.1145/1089733.1089734.

Knuth, D. E. und Merner, J. N. (1961). ALGOL 60 Confidential. *Communications of the ACM*, 4(6):268–272. DOI: 10.1145/366573.366599.

Koerber, B. und Peters, I.-R. (1995). Die Kurszeitung - Ein Einstieg in die informatische Bildung. *LOG IN*, 15(1).

Kohls, C. und Haug, S. (2008). Gemeinsam sind wir stark! *LOG IN*, (152).

Kölling, M. (2008). Using BlueJ to Introduce Programming. In *Reflections on the Teaching of Programming*, S. 98–115. Springer.

Kölling, M. (2010). The Greenfoot Programming Environment. *Transactions on Computing Education*, 10(4):14:1–14:21. DOI: 10.1145/1868358.1868361.

König, A. (2008). Kooperativ-Kollaborative Quelleninterpretation mit Wikis. *LOG IN*, (152).

Kultusministerkonferenz (KMK), Hrsg. (2009). *Empfehlung der Kultusministerkonferenz zur Stärkung der mathematisch-naturwissenschaftlich-technischen Bildung.*

Larman, C. und Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. *Computer*, 36:47–56. DOI: 10.1109/MC.2003.1204375.

Laughnan, J. (2004). The Organization of Inventing and Prototyping Activities with Children as Design Partners. In *Proceedings of the 2004 Conference on Interaction Design and Children: Building a Community*, S. 127–128. ACM. DOI: 10.1145/1017833.1017854.

Lehmann, E. (1996). Was tun mit Informatik-Freaks? *LOG IN*, 16(3).

Ljungblad, S. und Holmquist, L. E. (2007). Transfer Scenarios: Grounding Innovation with Marginal Practices. In *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, S. 737–746. ACM. DOI: 10.1145/1240624.1240738.

Loftus, C. und Ratcliffe, M. (2005). Extreme Programming Promotes Extreme Learning? In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '05, S. 311–315. ACM. DOI: 10.1145/1067445.1067531.

Maass, S. und Wiesner, H. (2006). Programmieren, Mathe und ein bisschen Hardware ... Wen lockt dies Bild der Informatik? *Informatik-Spektrum*, 29(2):125–132. DOI: 10.1007/s00287-006-0059-y.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., und Eastmond, E. (2010). The Scratch Programming Language and Environment. *Transactions on Computing Education*, 10(4):16:1–16:15. DOI: 10.1145/1868358.1868363.

Margolis, J. und Fisher, A. (2001). *Unlocking the Clubhouse: Women in Computing.* MIT Press.

Mattern, F. und Floerkemeier, C. (2010). Vom Internet der Computer zum Internet der Dinge. *Informatik-Spektrum*, 33(2):107–121.

McDowell, C., Werner, L., Bullock, H., und Fernald, J. (2003). The Impact of Pair Programming on Student Performance, Perception and Persistence. In *Proceedings of the 25th International Conference on Software Engineering*, S. 602–607. IEEE.

McKinney, D. und Denton, L. F. (2005). Affective Assessment of Team Skills in Agile CS1 Labs: the Good, the Bad, and the Ugly. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '05, S. 465–469. ACM. DOI: 10.1145/1047344.1047494.

McNiff, J. (2011). *All You Need to Know About Action Research*. Sage Publications.

Meerbaum-Salant, O. und Hazzan, O. (2010). An Agile Constructionist Mentoring Methodology for Software Projects in the High School. *Transactions on Computing Education*, 9(4):21:1–21:29. DOI: 10.1145.1656255.1656259.

Mendes, E., Al-Fakhri, L. B., und Luxton-Reilly, A. (2005). Investigating Pair-Programming in a 2nd-year Software Development and Design Computer Science Course. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, S. 296–300. ACM. DOI: 10.1145/1067445.1067526.

Merrill, D., Kalanithi, J., und Maes, P. (2007). Siftables: Towards Sensor Network User Interfaces. In *Proceedings of the First International Conference on Tangible and Embedded Interaction (TEI'07)*, S. 15–17. ACM.

MoMoTech (2011). Monitoring von Motivationskonzepten für den Techniknachwuchs. Report, acatech - Deutsche Akademie der Technikwissenschaften.

Moraveji, N., Li, J., Ding, J., O'Kelley, P., und Woolf, S. (2007). Comicboarding: Using Comics as Proxies for Participatory Design with Children. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, S. 1371–1374. ACM. DOI: 10.1145/1240624.1240832.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., und Balik, S. (2003). Improving the CS1 Experience with Pair Programming. *ACM SIGCSE Bulletin*, 35(1):359–362. DOI: 10.1145/611892.612006.

O'Malley, C. (1992). Designing Computer Systems to Support Peer Learning. *European Journal of Psychology of Education*, 7(4):339–352.

Osguthorpe, R. T. und Graham, C. R. (2003). Blended Learning Environments: Definitions and Directions. *Quarterly Review of Distance Education*, 4(3):227–233.

Otto, T. (2002). Projektunterricht - elektronisch unterstützt. *LOG IN*, (120).

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books.

Potter, L. E. C., von Hellens, L. A., und Nielsen, S. H. (2009). Childhood Interest in IT and the Choice of IT as a Career: The Experiences of a Group of IT Professionals. In *Proceedings of the Special Interest Group on Management Information System's 47th Annual Conference on Computer Personnel Research*, SIGMIS CPR '09, S. 33–40. ACM. DOI: 10.1145/1542130.1542138.

Pruitt, J. und Grudin, J. (2003). Personas: Practice and Theory. In *Proceedings of the 2003 Conference on Designing for User Experiences*, DUX '03, S. 1–15. ACM. DOI: 10.1145/997078.997089.

Rasmussen, B. und Håpnes, T. (1991). Excluding Women from the Technologies of the Future?: A Case Study of the Culture of Computer Science. *Futures*, 23(10):1107–1119. DOI: 10.1016/0016-3287(91)90075-D.

Rauch, H. (1994). Kooperatives Lernen in vernetzten Systemen. *LOG IN*, (5/6).

Read, J. C., Markopoulos, P., und Druin, A. (2011). A Community for Child Computer Interaction. `http://www.chi2011.org/communities/child-computer-interaction/CCI.pdf`.

Reich, B. H. und Nelson, K. M. (2003). In Their Own Words: CIO Visions About the Future of In-House IT Organizations. *SIGMIS Database*, 34(4):28–44. DOI: 10.1145/957758.957763.

Resnick, M. (2007). Sowing the Seeds for a More Creative Society. *Learning & Leading with Technology*, S. 18–22.

Rico, D. F. und Sayani, H. H. (2009). Use of Agile Methods in Software Engineering Education. In *Proceedings of the 2009 Agile Conference*, AGILE '09, S. 174–179. IEEE. DOI: 10.1109/AGILE.2009.13.

Rolf, A. (2008a). *Mikropolis 2010: Menschen, Computer, Internet in der globalen Gesellschaft*. Metropolis-Verlag.

Rolf, A. (2008b). Wo bleiben die Informatik-Studenten? `http://www.abendblatt.de/ratgeber/wissen/hochschule/article523932/Wo-bleiben-die-Informatik-Studenten.html`.

Romeike, R. (2007a). Animationen und Spiele gestalten - Ein kreativer Einstieg in die Informatik. *LOG IN*, (146/147).

Romeike, R. (2007b). Kriterien kreativen Informatikunterrichts. In Schubert, S., Hrsg., *Didaktik der Informatik in Theorie und Praxis 12. GI-Fachtagung Informatik und Schule - INFOS 2007*. S. 57–68. Gesellschaft für Informatik e.V. (GI).

Romeike, R. (2008). Sichtweisen einer Kreativen Informatik. In Brinda, T., Fothe, M., Hubwieser, P., und Schlüter, K., Hrsg., *Didaktik der Informatik - Aktuelle Forschungsergebnisse*, S. 129–138. Gesellschaft für Informatik e.V. (GI).

Romeike, R. (2010). Das bessere Werkzeug - Anmerkungen zur Diskussion Etoys vs. Scratch. *LOG IN*, (163/164).

Romeike, R. und Reichert, D. (2011). PicoCrickets als Zugang zur Informatik in der Grundschule. In *Informatik in Bildung und Beruf. 14. GI-Fachtagung Informatik und Schule - INFOS 2011*, S. 177–186. Gesellschaft für Informatik e.V. (GI).

Rosenthal, R. (1995). Critiquing Pygmalion: A 25-Year Perspective. *Current Directions in Psychological Science*, 4(6):171–172.

Rosenthal, R. und Jacobson, L. (1968). Pygmalion in the Classroom. *The Urban Review*, 3(1):16–20. 10.1007/BF02322211.

Roussou, M., Kavalieratou, E., und Doulgeridis, M. (2007). Children Designers in the Museum: Applying Participatory Design for the Development of an Art Education Program. In *Proceedings of the 6th International Conference on Interaction Design and Children*, S. 77–80. ACM. DOI: 10.1145/1297277.1297292.

Royce, W. W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. In *IEEE WESCON*, S. 328–339. IEEE.

Rüdiger, B. (1999). Von der traditionellen zur computergestützen Arbeit. *LOG IN*, (3/4).

Scaife, M., Rogers, Y., Aldrich, F., und Davies, M. (1997). Designing for or Designing with? Informant Design for Interactive Learning Environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, S. 343–350. ACM. DOI: 10.1145/258549.258789.

Schild, J., Walter, R., und Masuch, M. (2010). ABC-Sprints: Adapting Scrum to Academic Game Development Courses. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, FDG '10, S. 187–194. ACM. DOI: 10.1145/1822348.1822373.

Schmitz, M. und Thiele, O. (1999). Beispiele zu projektbezogener Teamarbeit. *LOG IN*, (5).

Schneider, J.-G. und Johnston, L. (2005). eXtreme Programming: Helpful or Harmful in Educating Undergraduates? *Journal of Systems and Software*, 74(2):121–132. DOI: 10.1016/j.jss.2003.09.025.

Schubert, K., Stuhldreier, G., und Wulf, V. (2011). come_IN: Interkulturelle Computerclubs zur Förderung von Integrationsprozessen. *Informatik-Spektrum*, 34(3):286–293.

Schubert, S. und Schwill, A. (2011). *Didaktik der Informatik*. Spektrum Akademischer Verlag.

Schulman, L. S. (1987). Knowledge and Teaching: Foundations of the New Reform. *Harvard Educational Review*, 57(1):1–21.

Schwaber, K. und Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall.

Simon, B. und Hanks, B. (2008). First-Year Students' Impressions of Pair Programming in CS1. *Journal on Educational Resources in Computing (JERIC)*, 7(4). DOI: 10.1145/1316450.1316455.

Soler-Adillon, J. und Parés, N. (2009). Interactive Slide: an Interactive Playground to Promote Physical Activity and Socialization of Children. In *Proceedings of the 27th International Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, S. 2407–2416. ACM. DOI: 10.1145/1520340.1520343.

Stanton, D., Bayon, V., Neale, H., Ghali, A., Benford, S., Cobb, S., Ingram, R., O'Malley, C., Wilson, J., und Pridmore, T. (2001). Classroom Collaboration in the Design of Tangible Interfaces for Storytelling. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '01, S. 482–489. ACM. DOI: 10.1145/365024.365322.

Starruß, I. (2010). *Synopse zum Informatikunterricht in Deutschland*. Bakkalaureat, Technische Universität Dresden.

Stoll, T., Thalheim, K., und Timmermann, B. (2008). Die Katze im Computer. *LOG IN*, (154/155).

Takeuchi, H. und Nonaka, I. (1986). New New Product Development Game. *Harvard Business Review*.

Teague, J. (1996). A Structured Review of Reasons for the Underrepresentation of Women in Computing. In *Proceedings of the 2nd Australasian Conference on Computer Science Education*, ACSE '97, S. 91–98. ACM. DOI: 10.1145/299359.299374.

Thomas, M. (2002). *Informatische Modellbildung - Modellieren von Modellen als ein zentrales Elemenet der Informatik für den allgemeinbildenden Schulunterricht*. Dissertation, Universität Potsdam.

Turkle, S. (1984). *The Second Self: Computers and the Human Spirit*. Simon & Schuster.

Universität Hamburg (2005). Kapazitäts- und Zulassungsberechnungen für Studiengänge der Universität Hamburg 2005/2006.

Universität Hamburg (2006). Kapazitäts- und Zulassungsberechnungen für Studiengänge der Universität Hamburg 2006/2007.

Universität Hamburg (2007). Kapazitäts- und Zulassungsberechnungen für Studiengänge der Universität Hamburg 2007/2008.

Universität Hamburg (2008). Kapazitäts- und Zulassungsberechnungen für Studiengänge der Universität Hamburg 2008/2009.

Universität Hamburg (2009). Kapazitäts- und Zulassungsberechnungen für Studiengänge der Universität Hamburg 2009/2010.

Universität Hamburg (2010). Kapazitäts- und Zulassungsberechnungen für Studiengänge der Universität Hamburg 2010/2011.

Utting, I., Cooper, S., Kölling, M., Maloney, J., und Resnick, M. (2010). Alice, Greenfoot, and Scratch – A Discussion. *Transactions on Computing Education*, 10(4):17:1–17:11. DOI: 10.1145/1868358.1868364.

Weigend, M. (2005). Extreme Programming im Klassenraum. `http://www.fernuni-hagen.de/schulinformatik/xp_weigend.pdf`.

Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, S. 94–104.

Werner, L. L., Hanks, B., und McDowell, C. (2004). Pair-Programming Helps Female Computer Science Students. *Journal on Educational Resources in Computing*, 4(1). DOI: 10.1145/1060071.1060075.

Wessner, M. (2004). Lerngruppen. In Haake, J., Schwabe, G., und Wessner, M., Hrsg., *CSCL-Kompendium*, S. 202–207. Oldenbourg Verlag.

Wharton, C., Rieman, J., Lewis, C., und Polson, P. (1994). The Cognitive Walkthrough Method: a Practitioner's Guide. S. 105–140. John Wiley & Sons.

Wilhelm, R., Hrsg. (1996). *Informatik - Grundlagen, Anwendungen, Perspektiven.* C.H. Beck.

Williams, L. und Upchurch, R. (2001). In Support of Student Pair-Programming. In *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education*, S. 327–331. ACM. DOI: 10.1145/364447.364614.

# Wissenschaftliche Veröffentlichungen des Autors

## Aus der Dissertation hervorgegangene Publikationen

Göttel, T. (2009a). Culturia: Intercultural Learning by Designing Games. In Turgeon, W. C., Hrsg., *Creativity and the Child: Interdisciplinary Perspectives*. The Inter-Disciplinary Press.

Göttel, T. (2009b). Virtual Sandbox: Adding Groupware Abilities to Scratch. In *Proceedings of the 8th International Conference on Interaction Design and Children*, S. 158–161. ACM. DOI: 10.1145/1822348.1822373.

Göttel, T. (2011a). Agiler Informatikunterricht: Soziale Aspekte der professionellen Softwareentwicklung im Schulunterricht erfolgreich erfahrbar machen. In *Informatik in Bildung und Beruf. 14. GI-Fachtagung Informatik und Schule - INFOS 2011*, S. 37–46. Gesellschaft für Informatik e.V. (GI).

Göttel, T. (2011b). Child-Personas and Agile Methods in Class to Focus on Social Aspects of Computer Science. In Workshop *Opportunities and Challenges when Designing and Developing with Kids @ School*. `http://workshops.icts.sbg.ac.at/KidsSchool/papers/7_Goettel_IDC11.pdf`.

Göttel, T. (2011c). Reviewing Children's Collaboration Practices in Storytelling Environments. In *Proceedings of the 10th International Conference on Interaction Design and Children*, S. 153–156. ACM. DOI: 10.1145/1999030.1999049.

# Weitere Publikationen während des Promotionszeitraums

Schild, J. und Göttel, T. (2010). Vorwort Workshop Game Design und Game Development in der Hochschulinformatik. In *Interaktive Kulturen - Workshop-Band Proceedings der Workshops der Mensch & Computer 2010*, S. 282.

Göttel, T. und Schild, J. (2011d). Creativity Room 5555: Evoking Creativity in Game Design amongst CS Students. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, S. 98–102. ACM. DOI: 10.1145/1999747.1999777.

Schild, J., Göttel, T. und Grimm, P. (2011e). Game Development Inhalte in der Hochschulinformatik. In *Workshopband Mensch & Computer 2011*, S. 358–390.

# Danksagung

Ich danke meinem Doktorvater Prof. Dr. Horst Oberquelle für die herzliche Betreuung und die stetige Ermutigung, eigene Themen zu verfolgen. Darüber hinaus bedanke ich mich bei meinem Gutachter Prof. Dr. Norbert Breier und den weiteren Mitgliedern der Prüfungskommission Prof. Dr. Ingrid Schirmer und Prof. Dr. Christopher Habel für den vorbildlichen Ablauf und die sehr angenehme Atmosphäre während der Schlussphase meiner Promotion. Ein Dankeschön an Hildegard Westermann, die alle organisatorischen Angelegenheiten stets freundlich und schnell bearbeitete.

Mein Dank gilt darüber hinaus meiner Familie und meinen Freunden. Sie geben mir verlässlichen Rückhalt und eröffnen mir stets neue Blickwinkel. Für die konstruktive Kritik zu meiner Arbeit danke ich insbesondere Ilse Göttel-Dauber, Katrin Böhme, Jonas Schild, Uwe Hahne und Sonja Kastner.

# Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Berlin, den 15. April 2012

Timo Göttel