Vermittlung von agiler Softwareentwicklung im Unterricht

Martin Kropp, FHNW, martin.kropp@fhnw.ch

Andreas Meier, ZHAW, meea@zhaw.ch

Zusammenfassung

Über den Hype hinaus, der um agile Softwareentwicklung entstanden ist, zeigen verschiedene Umfragen, dass dieses Vorgehen in der Praxis in verschiedener Hinsicht tatsächlich zu Verbesserungen in der Durchführung von Software-Projekten führt. Firmen, die agile Methoden einsetzen, geben an, dass sie seither zufriedener mit ihrem Entwicklungsprozess sind und insbesondere der Umgang mit Anforderungsänderungen sich wesentlich verbessert hat.

Andererseits sind entsprechend ausgebildete Fachleute jedoch Mangelware. In der Praxis sind deshalb Software Ingenieure mit Kompetenzen in agilen Methoden sehr gefragt.

Was bedeutet dies für die Software-Technik Ausbildung an Hochschulen? Was sind die speziellen Herausforderungen? Wie kann agile Software Entwicklung, die neben konkreten Techniken und Praktiken auf der individuellen Ebene, auch auf Team-Ebene und Werte-Ebene spezielle Anforderungen stellt, überhaupt vermittelt werden? Wie kann die Ausbildung von agiler Softwareentwicklung in die Ingenieur-Ausbildung integriert werden?

In diesem Artikel stellen wir unser Konzept zur Ausbildung von agilen Methoden an Hochschulen vor und berichten über unsere Erfahrungen als Dozierende.

Einleitung

Jüngste Umfragen zeigen, dass agile Software Entwicklungsmethoden in verschiedener Hinsicht zu wesentlichen Verbesserungen in der Durchführung von Software-Projekten führt [1,2]. Diese Erfolgsmeldungen tragen wesentlich dazu bei, dass agile Softwareentwicklungsmethoden in der IT-Industrie eine immer grössere Akzeptanz finden.

In der von den Autoren durchgeführten Studie (Swiss Agile Study) über den Einsatz von Software Entwicklungsmethoden in der Schweizer IT-Industrie [1] wurden diese Aussagen bestätigt. Die Studie liefert auch konkrete Zahlen über die Verbreitung, den Nutzen, den Einsatz von konkreten Praktiken, aber auch die Herausforderungen, die mit agilen Methoden einhergehen.

Um die Relevanz der agilen Methoden für die Ausbildung zu ermitteln, war für uns unter anderem die Beantwortung folgender Fragen wichtig:

1. Wie verbreitet ist agile Softwareentwicklung in der Praxis?
2. Haben agile Methoden wirklich Vorteile gegenüber den traditionellen, plan-getriebenen Methoden?
3. Was sind die entscheidenden Erfolgsfaktoren bei der agilen Softwareentwicklung?

Im nächsten Abschnitt werden wir die aus unserer Sicht für die Ausbildung wichtigsten Resultate der Studie vorstellen und danach einige Überlegungen und Konsequenzen für die Ausbildung an (Fach-) Hochschulen näher ausführen.

Nach einer Übersicht über die wesentlichen Elemente der agilen Methoden werden wir im Anschluss unsere Konzepte vorstellen. Wir zeigen auf, wie die erforderlichen Kompetenzen vermittelt bzw. von den Studierenden erlernt werden können.

Anschliessend berichten wir über den konkreten Umsetzungsstand und unsere Erfahrungen, die wir gemacht haben.

Den Abschluss bildet ein Ausblick über die Weiterentwicklung der Konzepte und deren Umsetzung.

Verbreitung und Nutzen von agilen Methoden

In der Swiss Agile Study wurden mehr als 1500 IT-Firmenmitglieder der teilnehmenden ICT-Verbände SwissICT, ICTnet und SWEN befragt, die Rücklaufquote betrug knapp 10%. Dabei wurden sowohl agil als auch nicht-agil arbeitende Unternehmen einbezogen. Von den teilnehmenden Unternehmen gaben 57% an, dass sie mit agilen Methoden entwickeln.

Auf die Frage, wie die agilen Methoden die Entwicklung beeinflusst hat, gaben die Unternehmen

zu allen befragten Aspekten an, dass sich diese verbessert (+) oder sogar sehr stark verbessert (++) haben (siehe Tabelle 1). Dabei sind insbesondere die Aspekte „Ability to manage changing Requirements“ (89%), „Development Process“ (80%) und „Time To Market“ (76%) zu nennen.

Aspect	--[1]	-	*o*	+	++
Time to market	**1**	**2**	**19**	**53**	**23**
Ability to manage changing priorities	**1**	**0**	**9**	**45**	**44**
Productivity	0	2	33	47	15
Software quality	0	2	45	35	16
Alignment between IT & business objectives	0	1	25	46	23
Project visibility	0	2	25	39	28
Risk management	0	5	32	42	17
Development process	**0**	**2**	**17**	**58**	**22**
Software maintainability / extensibility capability	0	7	55	23	12
Team morale	0	4	25	42	24
Development cost	1	12	52	22	7
Engineering discipline	0	4	42	42	9
Management of distributed teams	0	5	42	19	6
Requirements management	0	2	29	51	13

Tabelle 1. ***How has agile software development influenced the following aspects?***[2][3]

Interessant an den Ergebnissen in dieser Tabelle ist, dass mehr als die Hälfte der Unternehmen angeben, dass sich die Wartbarkeit der Software sowie die Kosten nicht verbessert haben. Aus Sicht der Ausbildung ist von Interesse, welche Praktiken und Techniken entscheidend zum Erfolg in agilen Projekten beitragen, da wir auf diese Techniken besonderes Augenmerk legen sollten.

Bei dieser Frage haben wir nach *Engineering Practices* wie z.B. Unit Testing und Continuous Integration sowie *Management Practices* wie Daily Standup, On-Site Customer und Planung unterschieden. Tabelle 2 gibt die detaillierten Resultate für die *Engineering Practices* wieder. Dabei geben die Spaltenwerte die Wichtigkeit für den Erfolg von „ganz unwichtig“ (--) bis „sehr wichtig“ (++) wieder. Als wichtige bzw. sehr wichtige *Engineering Practices* werden dabei insbesondere Kodierrichtlinien (82%), Unit Testing (76%) und Continuous Integration (70%) genannt.

Aspect	--	-	+	++
Behavior Driven Development (BDD)	21	24	17	8
Acceptance Test Driven Development (ATDD)	18	22	20	17
Test Driven Development (TDD)	10	16	29	30
Coding standards	**4**	**9**	**40**	**42**
Collective code ownership	12	12	33	30
Pair programming	20	32	25	18
Unit testing	**2**	**17**	**25**	**51**
Refactoring	3	25	35	28
Automated builds	8	14	26	40
Continuous integration	**4**	**16**	**32**	**38**
Automated acceptance testing	20	26	28	14
Continuous delivery	12	23	38	17

Tabelle 2. How important were the following agile engineering practices for your successful agile projects?[3]

Bei der Interpretation der Daten ist zu beachten, dass gewisse agile Praktiken wie z.B. TDD, BDD, ATDD auch bei agilen Unternehmen noch relativ wenig im Einsatz sind, wie sich bei der Umfrage gezeigt hat.

Bei den *Management Practices* wurden als wichtigste Aspekte Iterationsplanung (89%), User Stories (83%) und Release-Planung (77%) genannt, dicht gefolgt vom Einsatz eines Task Boards (74%).

Aspect	--	-	+	++
Release planning	**3**	**18**	**37**	**40**
Story mapping	4	26	41	21
On-site customer	11	27	33	24
Iteration planning	**0**	**7**	**36**	**53**
User stories	**0**	**10**	**40**	**43**
Daily standup	4	25	32	33
Taskboard	**2**	**20**	**45**	**29**
Burndown charts	11	38	22	24
Retrospective	5	28	34	30
Open work area	12	25	40	11
Kanban Pull System/Limited WIP	27	23	13	4

Tabelle 3. How important were the following agile management and planning practices for your successful agile projects?

Ein wichtiger Aspekt ist, dass die Zufriedenheit der Unternehmen mit den agilen Methoden insgesamt

[1] *Die Bewertungskriterien gehen von „Stark verschlechtert (--) bis „stark verbessert“(++)*

[2] *Sämtliche Angaben in Prozent; die fehlenden Prozent auf 100 gaben an “Weiss nicht”.*

[3] *Die Umfrage wurde auf Englisch durchgeführt, um alle Sprachregionen der Schweiz abdecken zu können, daher sind sowohl die Fragen als auch die Antwort-Optionen auf Englisch.*

deutlich höher ist als mit traditionellen plangetriebenen Vorgehen (84% vs. 62%).

Anforderungen an agile Ausbildung

Im Rahmen der Studie haben wir auch erhoben, ob agile Methoden überhaupt auf universitärer Stufe unterrichtet werden sollten, und wie die Kenntnisse der Bachelor- und Master-Absolventen von der Industrie beurteilt werden. Dazu hatten die Firmenvertreter der teilnehmenden IT-Firmen folgende Aussage zu bewerten:

„*Agile development should be an integral part of the Computer Science curriculum*".

Zur Auswahl standen folgende Antwort-Optionen von "Stimme vollständig zu" bis "Stimme gar nicht zu".

95% der teilnehmenden Firmen stimmen der Aussage zu („Stimme zu": 49%, „Stimme vollständig zu": 46%).

Andererseits geben die Unternehmen auf die Frage, ob die Studienabgänger auf Bachelor- bzw. Masterstufe genügend Kenntnisse in agilen Methoden mitbringen mit klarer Mehrheit an, dass dies nicht der Fall ist (Master-Stufe: 58%, Bachelor-Stufe: 68%).

Somit sollte der Anspruch der Praxis an die Hochschulen ernst genommen werden und die Ausbildung an die neuen Anforderungen angepasst werden. Dabei stellt sich die Frage, was die Hochschulen in der Ausbildung ändern müssen, um den neuen Anforderungen gerecht zu werden.

Dazu betrachten wir Anforderungen, die agile Methoden an den einzelnen aber auch an das Team und die Organisation stellt, auf drei verschiedenen Ebenen:

1. *Individuelle Ebene*: Hier geht es vornehmlich darum, welche agilen *Engineering Practices* ein Software Ingenieur beherrschen sollte: Clean Code, Test Driven Development, Automation, Craftsmanship, um nur einige Beispiele zu nennen.
2. *Team Ebene*: Die Team Ebene stellt vor allem Anforderungen im Bereich *Management Practices:* Selbstorganisierende Teams, Schätzen und Planen, Continuous Integration und Continuous Deployment, oder Pair Programming. Selbstorganisierende Teams wiederum verlangen eine hohe soziale Kompetenz der Teammitglieder, wie zum Beispiel hohe Kommunikationsfähigkeit.
3. *Werte Ebene*: Auf dieser Ebene wird implizit das Wertesystem der agilen Softwareentwicklung betrachtet; wie die agilen Werte wie Respekt, Offenheit, Ehrlichkeit, etc. vermittelt werden können, so dass die Absolventen „in der Wolle gefärbte" Software Ingenieure sind.

Bei der Vermittlung der Anforderungen beziehen wir uns schwerpunktmässig auf *Scrum* [7] und *eXtreme Programming (XP)* [8], da diese in der Schweiz (und auch international [2]) die am weitesten verbreiteten agilen Methoden sind. Ausserdem ergänzen sich diese Methoden aus unserer Sicht sehr gut, da XP eher den Schwerpunkt auf die *Engineering Practices* legt, während Scrum eher auf der Management Ebene anzusiedeln ist.

Erfahrungen mit agiler Entwicklung im Unterricht

Zum Thema der Integration von agilen Methoden in den Unterricht liegen verschiedene Erfahrungsberichte vor [20, 21]. Die Autoren selbst unterrichten seit über fünf Jahren zusammen die Vorlesung *Software Engineering and Architecture.* Diese Vorlesung ist Teil der Schweiz weiten gemeinsamen Masterausbildung (Master of Science in Engineering, MSE) der Schweizerischen Fachhochschulen [3]. Die gemeinsame Masterausbildung betrachten die Autoren als eigentlichen Glücksfall, da dadurch Dozierende von verschiedenen Hochschulen zusammen lehren und ihre Erfahrungen austauschen können.

Wir haben festgestellt, dass die Studierenden noch relativ wenig Wissen über agile Softwareentwicklungsmethoden mitbringen, wobei sich der Wissenstand in den letzten Jahren leicht verbessert hat, aber je nach Fachhochschule sehr unterschiedlich ist. Dafür beobachten wir jedoch, dass das Interesse der Studierenden an dem Thema umso grösser ist.

In unserer gemeinsamen Vorlesung haben wir den Fokus ganz gezielt auf agile Softwareentwicklung gelegt. Unter anderem unterrichten wir agile Software Entwicklungsmethoden wie Scrum, eXtreme Programming oder Kanban. Dabei haben wir die Erfahrung gemacht, dass viele Studierende das „Programmier-Handwerk", welches eine der Voraussetzungen für die erfolgreiche Durchführung agiler Projekte darstellt, nicht beherrschen resp. nie im Bachelor-Studiengang gelernt hatten. Unter „Programmier-Handwerk" verstehen wir das Beherrschen von agilen Praktiken wie zum Beispiel Clean Code, Refactoring, Test-Driven Development oder Continuous Integration.

Weiter halten die Autoren jeweils an ihrer Fachhochschule [4, 5] verschiedene Vorlesungen in Programmieren und Software Engineering auf Bachelor Stufe und haben damit begonnen, auch auf dieser Stufe agile Software Entwicklungsmethoden zu vermitteln.

Agile Software Entwicklung

Für ein besseres Verständnis der im Folgenden vorgestellten Studienkonzepte und Lehr- und Lernmethoden fassen wir hier die aus unserer Sicht wichtigsten Eigenschaften der agilen Softwareentwicklung zusammen.

In erster Linie sind dabei natürlich die Werte und Prinzipien des *Agile Manifesto* [6] zu nennen, die auf der einen Seite zwar noch keine konkreten Praktiken und Techniken vorgeben, aber mit Ihren Prinzipien doch klare Anforderungen an eine agile Vorgehensweise definieren.

Aus diesen Prinzipien hat insbesondere extreme Programming (XP) eine Anzahl ganz konkreter Programmierpraktiken abgeleitet, ohne die aus unserer Sicht eine agile Softwareentwicklung nicht möglich ist.

Auf Managementsicht gehört wohl Scrum zu den Vorreitern der agilen Methoden. Scrum definiert klare Regeln bzgl. Projekt- und Teamorganisation sowie dem Requirements-Management.

Neuere Ansätze wie Lean Software Development [9], die sich ebenfalls an den Prinzipien des *Agile Manifesto* orientieren, zeigen weitere, interessante Aspekte zur Umsetzung der agilen Software Entwicklung auf.

Die folgenden Tabellen fassen die aus unserer Sicht wichtigsten *Engineering* und *Management Practices* zusammen. Diese Einteilung wurde von uns für die Studie entwickelt, um die Verbreitung der genannten Praktiken in der Praxis zu ermitteln und hat sich dabei als sehr zweckmässig erwiesen; wir verwenden sie daher auch in diesem Artikel als Grundlage für die folgende Diskussion über die agile Ausbildung.

Tabelle 4 gibt die Verbreitung der, vor allem von XP definierten, konkreten Programmierpraktiken in den befragten agilen Unternehmen wieder, während Tabelle 5 die Verbreitung der Management Praktiken aufzeigt. Besonders erwähnenswert dabei ist, dass diese entsprechenden Werte bei plangetrieben arbeitenden Unternehmen zwischen 10%-30% tiefer liegen. Dies ist doch umso bemerkenswerter, als es sich bei vielen *Engineering Practices* eigentlich nicht um spezielle agile Praktiken, sondern um allgemeine *Best Practices* handelt. Es lässt sich also sagen, dass die Anwendung von agilen Methoden auch zur verstärkten Anwendung von allgemeingültigen Best Practices führt.

Engineering Practices	Verbreitung
Coding standards	75%
Unit testing	70%
Automated builds	63%
Continuous integration	57%
Refactoring	51%
Test Driven Development (TDD)	44%
Pair programming	31%
Collective code ownership	30%
Continuous delivery	30%
Automated acceptance testing	24%
Acceptance Test Driven Development (ATDD)	18%
Behavior Driven Development (BDD)	15%

Tabelle 4. Engineering Practices in der Praxis[4]

Management Practices	Verbreitung
Release planning	75%
Iteration planning	66%
User stories	65%
Daily standup	53%
Taskboard	46%
Retrospective	41%
Burndown charts	40%
Story mapping	35%
Open work area	27%
On-site customer	23%
Kanban Pull System/Limited WIP	11%

Tabelle 5. Management Practices in der Praxis[5].

Die Werte zeigen deutlich, dass auch bei agil entwickelnden Unternehmen erst relativ wenige der erforderlichen bzw. empfohlenen Praktiken eine breite Anwendung finden.

Studienkonzept für agile Methoden

Die Eigenschaften bzw. Anforderungen an eine agile Software Entwicklung lassen sich nicht alle auf die gleiche Art unterrichten, da sie unterschiedliche Kompetenzen ansprechen. Für die Entwicklung eines geeigneten Studienkonzeptes orientieren wir uns daher an den zuvor eingeführten drei Kompetenzebenen.

---

[4] *Den Firmen, die agile Methoden anwenden, wurde folgende Frage gestellt: „Which of the following engineering practices could be observed by someone visiting your company in the next month"?*

[5] *Den Firmen, die agile Methoden anwenden wurde folgende Frage gestellt: „Which of the following management and planning practices could be observed by someone visiting your company in the next month"?*

Die drei Vermittlungsebenen

Auf der *individuellen Ebene* werden insbesondere die handwerklichen Grundlagen vermittelt. Ohne das Beherrschen dieser fundamentalen Techniken ist eine agile Entwicklung aus unserer Sicht nicht möglich. Aus unserer Erfahrung sind die agilen Techniken der individuellen Ebene am einfachsten zu vermitteln. Dies liegt unserem Ermessen nach daran, dass jeder Studierende einzeln daran arbeiten kann.

Darauf aufbauend können dann die Techniken der agilen Entwicklung auf der *Team-Ebene* vermittelt und erworben werden, die sich vor allem aus den *Management Praktiken* zusammensetzen. Diese Aspekte lassen sich aus unserer Erfahrung heraus sehr gut in Gruppen- und Projektarbeiten vermitteln. Die Konzeption, Organisation und Durchführung solcher Arbeiten ist aufwendig.

Sozusagen die Spitze der Kompetenzen der Agilen Entwicklung bildet die *Werte-Ebene*. Diese agilen Wertehaltungen wie sie im *Agile Manifesto* [6] beschrieben werden, sind naturgemäss am schwierigsten zu vermitteln. Wir versuchen die agilen Werte punktuell immer wieder in den Unterricht einfliessen zu lassen – oder ganz bewusst auch vorzuleben.

Bachelor- und Master-Ausbildung

Aufgrund der sich schon früh abzeichnenden zunehmenden Bedeutung der agilen Entwicklungsmethoden haben wir begonnen die grundlegenden Methoden und Techniken der agilen Software Entwicklung in Form der genannten Management und Engineering Praktiken in das Bachelor-Programm zu integrieren [10,11]. Dies wird im Folgenden noch genauer erläutert.

Auf Master-Stufe behandeln wir in unserem gemeinsamen Kurs „Software Engineering and Architecture“ fortgeschrittene Themen der agilen Software Entwicklung wie Lean Development und Kanban, Incremental Software Design und Software Evolution.

Unterrichtsmethoden

Neben konventionellen Vorlesungen und Programmierübungen verwenden wir als weitere Unterrichtmethoden:

- Case Studies (Fremde / Eigene)
- Gruppenarbeiten
- Gastreferate mit Referenten aus der Software-Industrie
- Simulationen
- Medien: eBooks, Blogs, Internet.

Wichtig ist die Repetition auf allen Ebenen, d.h. regelmässiges Üben in verschiedenen Kontexten. Deshalb kommen die verschiedenen Praktiken und Techniken immer wieder in den verschiedenen Vorlesungen vor. Nicht isoliertes Unterrichten der einzelnen Praktiken steht im Vordergrund, sondern die Verankerung an mehreren Stellen in der Ausbildung. Das unterstützt insbesondere auch die Werte Ebene, in dem die Dozierenden selbst im Rahmen ihrer Möglichkeiten diese agilen Werte vorleben.

Beispiel Bachelor Programm

Obwohl an den beiden Fachhochschulen, auch was das Themengebiet Software Engineering angeht, unterschiedliche Studienpläne existieren, haben sich, unterstützt durch den intensiven Austausch, viele Gemeinsamkeiten in der Vermittlung der agilen Methoden ergeben.

Abbildung 1 zeigt als ein mögliches Beispiel einen

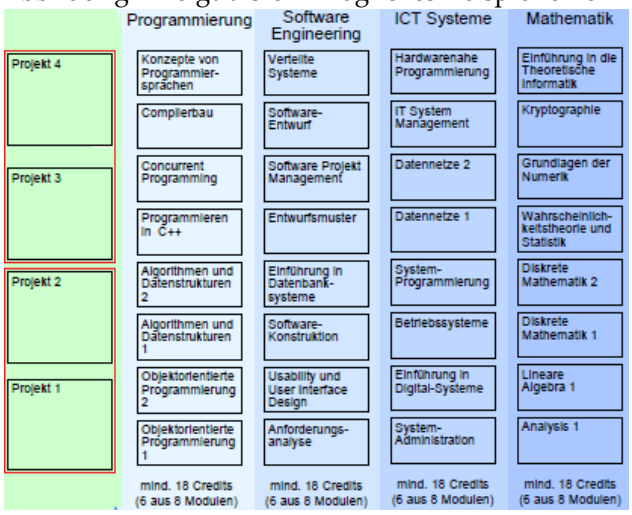

Abbildung 1. Grundstudium Informatik

Ausschnitt des Informatik Bachelor-Programms der Fachhochschule Nordwestschweiz.

Das fachliche Grundstudium im Bachelorstudiengang ist aufgeteilt in die vier Themenbereiche, Modulgruppen genannt, Programmierung, Software Engineering, sowie ICT Systeme und Mathematik. Jede Modulgruppe besteht aus je 8 Modulen, die spezielle Themen aus der jeweiligen Modulgruppe abdecken. Jedes Modul hat einen Umfang von 3 ECTS Punkten.

Begleitet wird die theoretische Ausbildung in den Modulen durch eine vom ersten Semester an beginnende Projektschiene, die sich bis in das 6. Semester fortsetzt. In dieser setzen die Studierenden in grossen Teams (7 bis 8 Studierende) und an realen, d.h. von externen Kunden in Auftrag gegebenen Projekten, die Theorie in die Praxis um. Dabei werden in den beiden Jahresprojekten der ersten 4 Semester unterschiedliche Schwerpunkte gesetzt.

Die explizite Ausbildung in agilen Methoden wird insbesondere in den Modulen Software Konstruktion im 2. Semester und Software Projekt Management im 3. Semester der Modulgruppe Software Engineering vorgenommen. In den Projekten und den sonstigen „programmierlastigen" Modulen wird darauf geachtet, dass auch hier die vermittelten agilen Praktiken zum Einsatz kommen.

In den folgenden Kapiteln beschreiben wir, wie wir an beiden Fachhochschulen die verschiedenen Praktiken und Techniken im Unterricht vermitteln.

Engineering Practices

Bei der Besprechung der Practices orientieren wir uns an der Liste aus Tabelle 4.

Coding Standards

Bereits im ersten Semester führen wir Coding Standards in der Einführungsvorlesung ins Programmieren ein. Wir kombinieren Coding Standards mit *Clean Code* [12], um so bei den Studierenden von Anfang an auch das Bewusstsein für eine „saubere" Programmierung zu wecken. Mit dem Einbezug von Clean Code haben wir sehr gute Erfahrungen gemacht, da dies den konkreten praktischen Nutzen von Coding Standards sichtbarer macht.

Von nicht zu unterschätzender Wichtigkeit ist bei der Einführung der Aspekt des Feedbacks. Das wird wie folgt gehandhabt:

Normalerweise gehört zu jeder Vorlesung eine Programmieraufgabe. Der Dozierende führt mit jedem Studierenden einen Code Review durch und gibt ein ausführliches Feedback. Das Feedback ist im Allgemeinen mündlich, kann aber auch schriftlich erfolgen.

Das Feedbackgeben durch den Dozierenden ist zwar mit einen grossem Aufwand verbunden, ist aus unserer Erfahrung aber entscheidend für den Lernerfolg.

Coding Standards müssen natürlich durch die verwendete integrierte Entwicklungsumgebung (IDE) unterstützt werden. Minimal sollte die IDE auch Unterstützung für einfache *Refactorings* wie *Rename Method* oder *Rename Variable* bieten.

Ebenfalls gute Erfahrungen haben wir mit Werkzeugen wie *Checkstyle* [16] gemacht, welches direkt in der IDE anzeigt, falls die Coding Standards verletzt werden.

Sehr positive Erfahrungen haben wir mit dem frühzeitigen Einsatz dieser Konzepte und Werkzeuge (z.B. im 2. Semester im Rahmen des Modul Software Konstruktion) zusammen mit *Continuous Integration* Umgebungen gemacht. Durch eine frühe Einführung werden die Studierenden animiert, diese Praktiken auch in ihren Projekten einzusetzen, was auch sehr häufig auf freiwilliger Basis geschieht, da sie die Vorteile solcher Umgebungen schon früh erkennen.

Unit Testing

Unit Testing wird im ersten Semester eingeführt. Dabei ist besonders wichtig, dass diese Praktik nicht nur in speziellen Modulen unterrichtet wird, sondern, dass auch sonstige programmierlastige Module Unit Tests einfordern oder vorgeben, um so z.B. die erfolgreiche Implementierung einer Programmieraufgabe zu überprüfen.

So können zum Beispiel im ersten Semester während der Einführung ins Programmieren Unit Tests bei Übungen vorgegeben werden. Die Aufgabe gilt als erfüllt, wenn das zu entwickelnde Programm alle Unit Test besteht. Die Studierenden lernen so spielend die Nützlichkeit von automatischen Tests kennen und erleben nebenbei, wie wertvoll Tests als Dokumentation sind.

Anschliessend wird das automatische Testen mit White-Box/Black-Box Tests und Äquivalenzklassen formal eingeführt. Die Studierenden sind dann in der Lage, einfache Testfälle selbstständig zu schreiben. In den höheren Semestern werden mit zunehmendem Know-how anspruchsvollere Testfälle mit Mock-Objekten entwickelt und verschiedene Testmuster eingeführt. Das automatische Testen sollte sich durch möglichst alle programmier-nahen Vorlesungen durchziehen.

Ein anderer erprobter Weg ist, dass die Studierenden schon vom ersten Semester an in den Programmier-Modulen das Schreiben von einfachen Unit-Tests mittels Unit Testing Frameworks kennenlernen und in den Projekten anwenden. Anschliessend erhalten sie im zweiten Semester im Modul Software Konstruktion eine vertiefte Einführung in das systematische Testen und fortgeschrittene Themen wie Mock Testing kennen, die sie dann wiederum in den Projektarbeiten für das fortgeschrittene Testen einsetzen können.

Refactoring

Test Driven Development (TDD) baut auf automatischen Unit Tests auf. Neben Kenntnissen in Unit Testing brauchen die Studierenden auch Kenntnisse in *Refactoring*. Mit anderen Worten sind gute Kenntnisse in Refactoring eine Voraussetzung für TDD.

Die einfachsten Code-Refactorings werden bereits am Anfang des Studiums zusammen mit den Coding Standards und der IDE Unterstützung eingeführt. Komplexere Design-Refactorings werden in den folgenden Semestern eingeführt. Dazu wird ein Katalog der verschiedenen Refactorings abgegeben und mit den Studierenden durchgearbeitet.

Ergänzt wird dieses Thema durch die Anwendung von komplexeren Refactorings wie Extract Class, Extract Method, Move Method anhand von Case Studies z.B. bei Überschreitungen von Schwellwerten bei Qualitätsmetriken.

Test Driven Development (TDD) und Refactoring

TDD ist eine anspruchsvolle Engineering Practice. Um TDD wirklich zu beherrschen, wäre es am Besten, wenn die Studierenden einen erfahrenen Software-Entwickler als Coach zur Seite hätten. Da dies im Rahmen einer Vorlesung oder Übungsstunde nicht wirklich realisierbar ist, sind alternative Vorgehensweisen nötig:

Bewährt haben sich folgende Alternativen:

- Case Studies
- Programming Katas
- Experten Videos
- Bücher und Fachartikel

Während den Studierenden mittels Experten Videos und Fachliteratur die konzeptionellen Aspekte des TDD vermittelt werden können, sollen Case Studies und Programming Katas den Studierenden dazu dienen, den TDD Ansatz praktisch zu üben und insbesondere dessen Einfluss auf Testbarkeit und Software Design selbst zu erfahren.

Automated Builds

Automated Builds werden mit Hilfe von *Ant-Skripts* und einem *Code Versioning System* (CVS) wie zum Beispiel SVN oder GIT eingeführt. Um den vollen Nutzen der Build-Automatisierung zu erfahren, bauen die Studierenden anhand einer konkreten Case Study über ein ganzes Semester ein Automatisierungsskript kontinuierlich aus, so dass das Kompilieren, Testen, Code Analysieren, als auch das Packaging der Software vollständig automatisiert durchgeführt werden kann.

Automated Builds sind eine Voraussetzung für Continuous Integration und werden im nächsten Kapitel genauer angeschaut.

Continuous Integration (CI)

Automated Builds und Continuous Integration werden mittels einer konkreten Fallstudie in einer Gruppenarbeit vermittelt und ebenfalls schon frühzeitig im Studium eingeführt (z.B. im Rahmen des Moduls Software Konstruktion im 2. Semester). In einem ersten Schritt wird das Beispielprojekt in ein Code Versioning System importiert. Im zweiten Schritt werden die Build-Skripts entwickelt, damit das Beispielprojekt automatisiert gebaut werden kann. Im dritten Schritt Konfigurieren die Studierenden den CI-Server (Jenkins). Im vierten Schritt werden verschiedene Jenkins-Erweiterungen besprochen und ausprobiert. Dabei werden verschiedene Metriken wie Code Coverage, Bindungsstärke von Klassen, Code Complexity, aber auch Lines of Code (LOC) behandelt.

Fortgeschrittene Metriken werden in höheren Semestern (Bewertung einer Software Architektur im 4. Semester Software Entwurf [17]) oder im Master-Studiengang (Technical Debit [18]) behandelt.

Es hat sich als zweckmässig erwiesen, den ganzen Bereich der Software Konstruktion in einer eigenen Vorlesung zusammenzufassen. Da Kenntnisse in Software Konstruktion eine wichtige Voraussetzung für erfolgreiche Projekt- und Bachelorarbeiten sind, sollte die Vorlesung bereits im zweiten oder dritten Semester durchgeführt werden.

Der nachhaltige Erfolg der frühzeitigen Einführung dieser Praktiken zeigt sich unter anderem daran, dass Studierende selbstständig eine solche Infrastruktur für ihre Projektarbeiten anfordern und einsetzen.

Pair Programming

Pair Programming üben wir nicht speziell. Es wird zusammen mit den anderen *Engineering Practices* eingeführt und die Studierenden werden ermuntert, Pair Programming bei Gelegenheit gezielt einzusetzen und für sich zu entscheiden, ob es für sie eine Option darstellt oder nicht.

Wir haben die Beobachtung gemacht, dass die meisten Studierenden ganz automatisch eine Form von Pair Programming einsetzten. Sehr oft setzten sich zwei Studierende zusammen an einen Computer und lösen eine Programmieraufgabe gemeinsam. Es scheint also so etwas wie eine natürliche Vorgehensweise beim Programmieren zu sein.

Automated Acceptance Testing

Auf Automated Acceptance Testing gehen wir ebenfalls im Rahmen der Vorlesung Software Konstruktion ein. Es wird ein Überblick über das Konzept anhand des Test Frameworks Fit [13] vermittelt und Akzeptanztests geschrieben. Diese Tests werden auch in den CI-Build Prozess integriert.

Collective Code Ownership

Ebenso wie Pair Programming wird auch das Thema der Collective Code Ownership nicht explizit unterrichtet. Die Studierenden werden jedoch durch Verwendung von Versionskontrollsystemen, Anwendung von Coding Standards und Wechseln der Verantwortlichkeiten in den Projekten dazu ermuntert, diese Praktik auszuprobieren.

Fortgeschrittene Engineering Practices

Die weiteren *Engineering Practices* wie Continuous delivery, Acceptance Test Driven Development (ATDD) oder Behavior Driven Development (BDD) sprengen unserer Ansicht nach den Rahmen eines Bachelor Studiums. Denkbar, und aus unserer Sicht auch sinnvoll, wäre es, diese weiteren *Engineering Practices* zukünftig im Master Studium zu behandeln, da wir davon ausgehen, dass diese Praktiken in Zukunft an Bedeutung gewinnen werden.

Management Practices

Wie aus Tabelle 5 zu entnehmen ist, sind bei den *Management Practices* diejenigen am weitesten verbreitet, welche sich direkt mit der Projektplanung befassen. Nicht unerwartet wird die Planung vorwiegend unter Verwendung von User Stories gemacht. Überraschenderweise sind Retrospektiven in der Praxis, trotz deren unbestritten hohem Nutzen, nicht sehr weit verbreitet.

Die *Management Practices* betreffen vorwiegend die Team Ebene. Teamarbeiten sind im Gegensatz zu den *Engineering Practices*, welche vorwiegend die Individuelle Ebene betreffen, nicht einfach in den Unterricht zu integrieren. Die meisten *Management Practices* können nur sinnvoll in einem grösseren Team im Rahmen eines „richtigen" Projekts sinnvoll geübt werden. Oft ist es aber nicht zweckmässig oder aus zeitlichen Gründen unmöglich, ein solches Projekt durchzuführen. In diesem Fall kann die Situation mit einem der agilen Games wie zum Beispiel dem XP-Game [14] oder Scrum-City-Game [15] simuliert werden.

Scrum

Als agile Projektmethode verwenden wir Scrum. Die verschiedenen Studien zeigen, dass Scrum am weitesten verbreitet ist. Scrum bietet auch den Vorteil, dass die Anzahl der *roles*, *events* und *artifacts* klein ist.

Wir haben die Erfahrung gemacht, dass die Funktionsweise von Scrum schnell im Unterricht erklärt werden kann. Der schwierige Teil kommt bei der Einführung. Wenn die Studierenden Scrum (oder eine andere Projektmethode) wirklich anwenden sollen. Wie kann im Unterricht ein Klima geschaffen werden, so dass die Studierenden sich wie in einem „echten" Projekt fühlen? Wenn sie das Gefühl haben, dass es sich um ein „Alibi"-Projekt handelt, setzten sie sich nicht ein und profitieren entsprechend wenig.

XP-Game

Das XP-Game [14] simuliert ein agiles Projekt. In drei bis vier Stunden werden drei Iterationen mit Schätzen der Tasks, Planung der Iterationen, Implementation und Abnahme der Tasks sowie einer Retrospektive nach jeder Iteration durchgespielt. Die Planung und Retrospektiven werden „richtig" gemacht, die Implementationen dauern jeweils nur zwei Minuten. In diesen zwei Minuten kann natürlich kein Code entwickelt werden. Anstelle werden kurze, einfache Tasks, wie zum Beispiel ein Kartenhaus bauen oder etwas im Kopf ausrechnen, durchgeführt.

Das XP-Game wurde bis jetzt vier Mal mit je ca. 25 Studierende im Master-Studiengang im Rahmen des Moduls „Software Engineering and Architecture" (3 ECTS) [19] durchgeführt.

Interessant sind dabei folgende Beobachtungen:

- Die erste Iteration ist typischerweise sehr chaotisch. Die Schätzungen sind schlecht und die Selbstorganisation der Teams funktioniert nicht.
- Nach der Retrospektive am Anschluss an die erste Iteration nimmt die Selbstorganisation der Teams merklich zu. Ab der zweiten Iteration funktionieren die Teams, die Schätzungen werden viel besser.
- Die Velocity wird erstaunlich konstant.
- Der Lerneffekt in dieser kurzen Zeit ist sehr gross. Die Studierenden sind aktiv am arbeiten.
- Das XP-Game macht grossen Spass!

Das Feedback der Studierenden ist durchwegs positiv. Die Studierenden schätzen es insbesondere auf eine spielerische Art einen grossen Lerneffekt zu erzielen.

Scrum City Game

Ein alternativer Ansatz um agile Management Praktiken zu vermitteln und, insbesondere, zu erfahren, ist das Scrum Lego City Game [15]. Hier bekommen die einzelnen Teams die Aufgabe gestellt in einem Mini-Projekt während vier 10 minütigen Sprints eine Stadt aus Lego-Bausteinen nach vorgegeben User Stories zu realisieren.

Wir führen das Scrum Lego City Game in Gruppen von 6 bis 7 Studierenden in leicht abgewandelter Form über mehrere Wochen durch:

- An Stelle von (teuren) Lego-Bausätzen wird die Stadt aus Karton und Papier mit Farbstiften Scheren und Klebstoff gebaut.
- Das Schreiben der User Stories ist Teil der Aufgabenstellung. Um zu verhindern, dass die Entwickler ihre eigenen Anforderungen schreiben, bekommt der Product Owner (PO) eines Teams die Entwickler eines anderen Teams als Benutzer zugewiesen und definiert mit diesen die Anforderungen für „sein" Team.

- Auch die Priorisierung der User Stories durch den PO und das Schätzen durch das Entwicklerteam gehören zur Aufgabe.

Danach werden in insgesamt 3-4 Sprints die Aktivitäten wie Sprint-Planung, Task Breakdown, Sprint Review, Retrospektive, Arbeiten am Scrum Board in Mini-Sprints von 10 Minuten Dauer durchlaufen. Aufgrund der kurzen Sprints entfällt der Daily Scrum.

Die dabei gemachten Erfahrungen decken sich durchwegs mit jenen des XP Games. Ein weiterer wichtiger Aspekt ist die Einprägsamkeit der verschiedenen Konzepte durch das konkrete Anwenden in dieser Spielform.

Entwicklung eines Computerspiels mit Scrum

In einem neuen Software Engineering Modul im 5. Semester wird ein anderer Weg beschritten. Die Studierenden sollen die Individuelle-, Team- und Werte-Ebenen während einem Semester möglichst intensiv erfahren und verinnerlichen. Die Vorlesung ist als eigentlicher „Crash"-Kurs für „agile Software Engineering" konzipiert.

In der ersten Hälfte des Semesters liegt der Fokus auf der Individuellen Ebene. Es gibt eine ausführliche Einführung in die *Engineering Practices* von eXtreme Programming: Coding Standards, Unit Testing, Continuous Integration (CI), etc. Die Studierenden werden mit dem automatischen Build Process bekannt gemacht und nach der Hälfte des Semesters haben sie bereits einen kompletten CI-Server in Betrieb genommen.

In der zweiten Hälfte liegt der Fokus auf der Team Ebene. Die Studierenden sollen in Gruppen von sechs bis acht Mitgliedern während den restlichen sieben Wochen ein eigenes 2D-Computerspiel entwickeln. Dazu bekommen sie eine ausführliche Einführung in Scrum sowie in agiler Schätzung und Planung.

Die Studierenden schreiben die User Stories, schätzen die Tasks und planen die Iterationen. Eine Iteration dauert nur eine Woche und der Umfang ist dementsprechend klein. Längere Iterationen sind nicht zweckmässig, da das Projekt mit nur sieben Wochen extrem kurz ist.

Die Dozierenden nehmen bei jeder Gruppe am wöchentlichen Sprint-Meeting teil und unterstützen die einzelnen Teams. Es wird grosser Wert auf die Planung und Retrospektive gelegt.

Die Dozierenden coachen die einzelnen Teams und geben Unterstützung bei auftretenden Schwierigkeiten. Sie versuchen auch, den Teams einen „Spiegel" vorzuhalten, damit sie ihre Entscheidungen reflektieren und dadurch die Konsequenzen der getroffenen (und natürlich auch der nicht getroffenen) Entscheidungen besser verstehen.

Der Wert dieses Scrum-Projekts besteht in der Kombination der *Engineering Practices* mit den *Management Practices* in einer realistischen Projektsituation. Während des Projekts gibt es auch oft Gelegenheit für den Coach, die Werte der agilen Softwareentwicklung einfliessen zu lassen.

Projekt- und Bachelorarbeiten

In den Projekt- und Bachelorarbeiten ist es den einzelnen Dozierenden überlassen, ob sie die Aufgabe mit einer agilen Projektmethode lösen lassen oder nicht.

Vor allem in der sogenannten „Projektwoche", in der die Projektteams einmal pro Semester eine ganze Woche am Stück an ihren Projekten arbeiten können, wird insbesondere das Taskboard für die intensive Teamarbeit genutzt.

Bei externen Projekten ist es eine Herausforderung auch die externen Auftraggeber für das agile Vorgehen zu überzeugen, da dies ein grösseres Engagement des Kunden erfordert. Die Erfahrungen mit solchen Projekten sind jedoch mehrheitlich positiv.

Eine Schwierigkeit, Projektarbeiten nach agilem Vorgehen zu organisieren, ist, dass in diesen Arbeiten die Teams aus Zeitgründen in der Regel nur einmal pro Woche gemeinsam am Projekt arbeiten können, sonst eher getrennt (z.B. Zuhause). Ein „Daily Standup" ist daher in der Regel nicht möglich. Die Studierenden machen daher eher ein Meeting, in dem sie sich gegenseitig auf den neuesten Stand bringen. Statt einer Pinnwand als Scrum Board, werden häufig digitale Boards eingesetzt, die jedoch nicht so einfach zu bedienen sind wie Pinnwände und nicht die gleiche Flexibilität aufweisen. Daher werden diese oft nicht mit der notwendigen Sorgfalt verwaltet.

Erfahrungen

Welche Erfahrungen haben wir nun mit dem Unterrichten von agilen Methoden gemacht, und wie wird dieses Vorgehen von den Studierenden aufgenommen?

Unsere Erfahrungen sind durchwegs positiv, auch wenn noch einige Herausforderungen bestehen. Die Studierenden sind sehr offen gegenüber den agilen Methoden, da sie den Fokus stärker auf die eigentlichen Interessen der Studierenden, die Konstruktion der Software durch Design und Programmierung, legen. Das Feedback der Studierenden ist entsprechend ermutigend.

Gerade auch der Nutzen von Test-getriebener Entwicklung und der Automatisierung mittels CI Umgebungen wird für die Studierenden sehr schnell

ersichtlich und, erfreulicherweise, von diesen dann selbst in Projektarbeiten eingesetzt.

Unterschätzt wird häufig das disziplinierte Vorgehen, welches agile Methoden vom gesamten Projektteam z.B. bei der iterativen Planung und der Software Qualität, einfordern. Hier gilt es sicherlich von der Ausbildungsseite her, ein Augenmerk darauf zu haben.

Die Simulationen von agilem Vorgehen mittels Games (XP, Scrum) hat sich sehr bewährt. In den Games „erfahren" die Studierenden wichtige agile Konzepte wie selbst-organisierte Teams und häufige Auslieferungen in sehr intensiven Mini-Projekten, die sich entsprechend einprägen.

Im Hinblick darauf, dass vor allem die agilen Werte, aber auch deren Praktiken nicht durch einmaliges Vermitteln sondern durch Vorleben und ständige Anwendung erlernt werden müssen, ist es wichtig, dass diese Praktiken auch in den sonstigen Programmiermodulen zum Einsatz kommen.

Die neuen zusätzlichen Anforderungen an die Dozierenden sind dabei nicht zu unterschätzen. Die Dozierenden der entsprechenden Module müssen sich die notwendigen Grundlagen aneignen. Die Vorlesungen müssen entsprechend angepasst, und die agilen Praktiken in die eigenen Module intergiert werden. Der höhere Aufwand der Dozierenden ist sicher eine Herausforderung, welcher aber aus unserer Sicht durch das gute Resultat mehr als gerechtfertigt ist.

Ein weiterer wichtiger Vorteil der Vermittlung der agilen Methoden ist, dass die vor allem von XP propagierten allgemeinen Best Practices viel stärker ins Bewusstsein der Studierenden rücken und auch eine stärkere Akzeptanz erfahren, da mit jeder Iteration eine Qualitätskontrolle stattfindet.

Zusammenfassung und Ausblick

Wie unsere und auch internationale Studien zeigen, sind agile Software Entwicklungsmethoden in der Praxis sehr stark im Kommen. In der Ingenieur-Ausbildung müssen wir uns den geänderten Kompetenzanforderungen stellen und unsere Ausbildungskonzepte entsprechend anpassen und erweitern.

Im vorliegenden Artikel haben wir beschrieben, wie wir die Vermittlung von agilen Softwareentwicklungsmethoden an unseren Fachhochschulen integriert haben. Die Erfahrungen damit sind sehr positiv und die erzielten Erfolge stimmen zuversichtlich. Erste Feedbacks aus der Praxis über die Kompetenzen der Abgänger bestätigen unseren eingeschlagenen Weg.

Eine generelle Herausforderung bleibt, die agilen Konzepte nicht nur in einzelnen Modulen zu unterrichten, sondern diese in möglichst vielen programmier-nahen Modulen anzuwenden und zu vermitteln, was generell ein Wechsel des Unterrichtskonzept der isolierten Fachmodule hin zu einem integrativem Gesamtstudienkonzept bedeutet.

Auf Master-Stufe wollen wir die Vermittlung fortgeschrittenen *Engineering* und *Management Practices* weiter ausbauen, da diese in Zukunft eine zunehmende Bedeutung erlangen werden.

Zum Schluss sei angemerkt, dass auch die agilen Softwareentwicklungsmethoden nicht das Ende der Entwicklung der Softwareprozesse darstellen. Die Informatik ist immer noch eine sehr junge Disziplin, deren Entwicklungsprozessmodelle sich stetig weiterentwickeln werden. Aus Sicht der Ingenieur-Ausbildung ist es daher zentral, sich abzeichnende Entwicklungen in der Ausbildung vorwegzunehmen und damit einen Beitrag dazu zu leisten, dass moderne Entwicklungsmethoden ihren Einzug in die Praxis halten.

Referenzen

[1] Kropp, M., Meier, A. Swiss Agile Study - Einsatz und Nutzen von Agilen Methoden in der Schweiz. www.swissagilestudy.ch, 2012 (Bericht in Bearbeitung).

[2] Version One. State of Agile Development Survey results. http://www.versionone.com/state_of_agile_development_survey/11/, 20.10.2012

[3] Software Engineering and Architecture. http://www.msengineering.ch/fileadmin/user_upload/modulbeschreibungen/de/TSM_SoftwEng_de.pdf, 29.10.2012.

[4] http://www.zhaw.ch/, 29.10.2012

[5] http://www.fhnw.ch/technik, 29.10.2012

[6] Agile Manifesto. http://agilemanifesto.org/, 29.10.2012.

[7] K. Schwaber, M. Beedle. Agile Software Development with Scrum. Prentice Hall, 2001.

[8] K. Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley, 2009

[9] M. und L. Poppendieck. Lean Software Development: An Agile Toolkit. Addison-Wesley, 2003.

[10] S. Hauser, M. Kropp. Software Projekt Management. http://web.fhnw.ch/plattformen/spm/ , 29.10.12

[11] Ch. Denzler, M. Kropp. Software Construction. http://web.fhnw.ch/plattformen/swc. 29.10.2012

[12] R.C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall, 2008.

[13] R. Mugridge, W. Cunningham. Fit for Developing Software: Framework for Integrated Tests. Prentice Hall, 2005.

[14] XP Game. XP-Game, http://www.xpgame.org, 29.10.2012

[15] Scrum Lego City Game. http://www.agile42.com/en/training/scrum-lego-city, 29.10.2012

[16] Checkstyle Homepage. http://checkstyle.sourceforge.net/, 31.10.2012

[17] Ch. Denzler. http://web.fhnw.ch/plattformen/swent, 31.10.2012

[18] W. Cunningham. The WyCash Portfolio Management System. OOPLSA 1992.

[19] Modul Software Engineering & Architecture, Master Of Science of Engineering, http://www.msengineering.ch/fileadmin/user_upload/modulbeschreibungen/de/TSM_SoftwEng_de.pdf, 18.12.2012

[20] D.F. Rico; H.H. Sayani. "Use of Agile Methods in Software Engineering Education," Agile Conference, 2009. AGILE '09. , vol., no., pp.174-179, 24-28 Aug. 2009.

[21] A. Shukla; L. Williams. "Adapting extreme programming for a core software engineering course ," Software Engineering Education and Training, 2002. (CSEE&T 2002). Proceedings. 15th Conference on , vol., no., pp.184-191, 2002.