Iterativ-inkrementelle Vermittlung von Software-Engineering-Wissen

Veronika Thurner, Hochschule München
thurner@hm.edu

Zusammenfassung

Viele Ansätze der Software Engineering Lehre behandeln in sequenzieller Abfolge die einzelnen Wissensbereiche nacheinander. Sie verlaufen damit analog zum Wasserfallmodell, bei dem ebenfalls einzelne Disziplinen nacheinander erschöpfend abgearbeitet werden, wobei die Sinnhaftigkeit und der Beitrag der dabei erzielten Zwischenergebnisse zum Gesamtergebnis oft erst relativ spät im Projekt erkennbar wird.

In der Praxis komplexer Software Engineering Projekte haben sich heute iterativ-inkrementelle bzw. agile Ansätze gegenüber dem Wasserfallmodell durchgesetzt. Dabei wird in kleinen in sich abgeschlossenen Zyklen, in vielen überschaubaren Schritten und mit kurzen Feedback-Zyklen nach und nach ein System weiterentwickelt.

Der hier vorgestellte Lehr-/Lernansatz greift diese Idee auf und überträgt sie auf die Vermittlung von Software Engineering Wissen. Dabei wird von der ersten Lehreinheit an mit einer inkrementellen Folge von Beispielsystemen gearbeitet, die quasi bei null anfängt und sukzessive immer komplexer wird. Jedes der enthaltenen Systeme spiegelt dabei eine kleine, aber vollständige Iteration durch den Software Life Cycle wider, von der Anforderungsskizze über Entwurf und Implementierung bis hin zum Test.

Motivation

Software Engineering ist heute eine komplexe und sehr vielschichtige Disziplin. Entsprechend hoch sind die Anforderungen, die an ihre Ausführenden gestellt werden. So sind beispielsweise bereits für die Bewältigung kleiner, überschaubarer Projekte Kenntnisse und Kompetenzen in so unterschiedlichen Bereichen wie Analyse oder Implementierung erforderlich, verbunden mit Modellierungssprachen, Architekturprinzipien und aktuellen Frameworks für die konkrete Umsetzung. Neben diesen eher technischen Kerndisziplinen und Wissensbereichen werden darüber hinaus Fähigkeiten in Projektmanagement, Vorgehensmodellen etc. benötigt, sowie diverse grundlegende überfachliche Kompetenzen (Selbst-, Sozial- und Methodenkompetenzen).

Einen guten Überblick über das Lernspektrum für angehende Softwareingenieure vermittelt der Software Engineering Body of Knowledge (Abran u. a., 2005), der nach den 11 identifizierten Wissensbereichen der Version von 2004 in der aktuell entwickelten Version 3 auf 15 Wissensbereiche ausgebaut werden wird (IEEE, 2012). Diese Erweiterung des SWEBOK ist eines von vielen Indizien dafür, dass mit einer gravierenden Vereinfachung des Software Engineerings bis auf Weiteres erst mal nicht zu rechnen ist.

Den Lehrenden stellen der Umfang und die Komplexität der zu vermittelnden Domäne vor zwei zentrale Fragen. Die eine, angesichts der in der Regel streng begrenzten verfügbaren Ausbildungszeit unvermeidliche lauet:

- Was lasse ich weg?

Oder, positiver formuliert:

- Was ist die Essenz, die ich vermitteln muss/will?

Die andere, bei diesem Meer von untereinander abhängigen Wissensbereichen ähnlich drängende Frage ist:

- Wo fange ich an?

Zur ersten Frage existieren bereits diverse allgemeinere und konkretere Überlegungen, welche die zugrunde liegende Problematik zwar vielleicht noch nicht endgültig lösen, aber zumindest Handlungsspielräume aufzeigen (Lehner, 2009).

Systematische Überlegungen zur zweiten Frage nach dem richtigen Einstiegspunkt und einer sinnvollen Vermittlungs- bzw. Lernreihenfolge stehen im Fokus der vorliegenden Arbeit.

Hierfür wird zunächst die aktuelle Verteilung der Lerninhalte auf die einzelnen Fächer im Software-Engineering-Lernpfad dargestellt und beleuchtet, welche Probleme sich in der Lehr-/Lernpraxis aus der derzeit etablierten, am Wasserfallmodell orientierten Reihenfolge der inhaltlichen Vermittlung ergeben. Darauf aufbauend wird kurz die Idee vorgestellt, die Lerninhalte des Software Engineerings nach einem iterativen, inkrementellen Prozess zu vermitteln. Anschließend wird das Beispiel erläutert, anhand dessen die einzelnen Lerninhalte schrittweise eingeführt und von den Studierenden in die Praxis umgesetzt werden. Eine Analyse der ersten mit diesem Lehrkonzept gewonnenen Erfahrungen runden die Arbeit ab.

Problemstellung

Den groben Rahmen der Antwort auf die „Wo fange ich an?“-Frage definieren im Hochschul-Kontext in der Regel die Studienpläne.

Einbettung in das Curriculum

Wir betrachten hier als konkretes Beispiel den Studiengang Bachelor Wirtschaftsinformatik an der Hochschule München, der für den Bereich Software Engineering den folgenden Ausbildungspfad vorsieht:

- 1. Semester: Softwareentwicklung 1
  Einführung in das objektorientierte Programmierparadigma und die Grundzüge der Programmierung am Beispiel der Sprache Java
- 2. Semester: Softwareentwicklung 2
  Vertiefung der Programmierkenntnisse und Einsatz fortgeschrittener objektorientierter Programmierkonzepte am Beispiel der Sprache Java
- 3. Semester: Software Engineering 1
  Objektorientierte Analyse und Entwurf mit UML, Architekturauswahl, Aufwandsschätzung, Qualitäts- und Projektmanagement
- 4. Semester: Software Engineering 2
  Werkzeuge zur Automatisierung der Entwicklung, modellgetriebene Entwicklung, Konfigurationsmanagement, Verifikation und Test, Softwarearchitekturen, Prozessmodelle und agile Vorgehensmodelle

Jede dieser Veranstaltungen ist verpflichtend und umfasst je zwei Semesterwochenstunden seminaristischen Unterricht und zwei Semesterwochenstunden Praktikum. An den Praktikumsgruppen nehmen in der Regel ca. 20 Studierende teil. Der seminaristische Unterricht ist auf ca. 40–50 Studierende ausgelegt, wobei hier bedingt durch die aktuellen starken Jahrgänge zum Teil erheblich nach oben abgewichen werden muss.

Ergänzt werden diese Pflichtfächer durch ein umfangreiches Angebot an Wahlfächern, deren Spektrum von Personalführung bis Webtechniken reicht und die ab dem 4. Semester besucht werden können.

Auftretende Schwierigkeiten

Obiger Studienplan legt also fest, dass nach einer Grundausbildung in objektorientierter Programmierung in den Software-Engineering-Veranstaltungen zunächst im 3. Semester die frühen, eher modellierungsorientierten Disziplinen sowie Management-Themen behandelt werden, während die eher technischen, implementierungsnahen Disziplinen sowie Vorgehensmodelle erst im 4. Semester auf der Tagesordnung stehen.

Diese Aufteilung bewirkt, dass die Studierenden also ein Semster lang Anforderungen erfassen und analysieren, daraus Entwürfe ableiten und all das mit umfassenden UML-Modellen konkretisieren. Wie sich die dabei gefällten Entscheidungen und erstellten Spezifikationen letztendlich auf den Quelltext und das zu entwickelnde System auswirken bleibt dabei zunächst eine rein theoretische Überlegung, da der Round Trip zur technischen Umsetzung erst ein Semester später stattfindet.

Selbst wenn die Software-Engineering-Lehrveranstaltungen durchgängig über zwei Semester hinweg konzipiert und durchgeführt werden bleibt der große zeitliche Abstand zwischen Modellierung und Implementierung problematisch. Letztlich dauert es insgesamt zu lange, bis in den Köpfen der Studierenden ein rundes Bild der gesamten Disziplinen im Software Life Cycle entsteht, sodass Zusammenhänge und Abhängigkeiten zwischen den einzelnen Techniken und Disziplinen oft erst sehr spät erkannt werden. Dadurch sinkt nicht nur die Lernintensität, sondern auch der Spaßfaktor, da der Nutzen vieler Inhalte erst gegen Ende des Entwicklungszyklus deutlich wird.

Falls zwischen dem dritten und vierten Semester dann auch noch der/die Dozierende wechselt und im Praktikum ein anderes Anwendungsbeispiel verwendet kommt der/die Studierende unter Umständen überhaupt nicht an den Punkt, an dem selbsterstellte Spezifikationen zu einem laufenden System umgesetzt werden müssen. Dadurch fehlt ein sehr wesentlicher Erkenntnisschritt im gesamten Lernprozess, da die Studierenden nicht unmittelbar an der eigenen Arbeit erfahren, wie sich ihre bei der Modellierung gefällten Entscheidungen bei der Implementierung auswirken und wieviel Mehraufwand ggf. erforderlich ist, um Modellierungsfehler auszubügeln, die ihren Ursprung in Anforderungsanalyse und Entwurf haben, aber erst bei der Implementierung erkannt werden.

In der aktuellen Lehrpraxis wird daher mitunter bereits im modellierungslastigen Semester zusätzlich zur Erstellung eines komplett ausmodellierten Pflichtenheftes nebst vollständigen Entwurfsmodellen wenigstens eine rudimentäre Umsetzung von Teilen der Systemfunktionalität gefordert, auch wenn ein großer Teil der dafür benötigten Kenntnisse erst ein Semester später systematisch in der Lehrveranstaltung behandelt wird. Insbesondere implementierungsschwächere Studierende sind damit in der Regel völlig überfordert.

Die potenziellen Programmiergurus schaffen es dagegen bis zu einem gewissen Grad, sich durch entsprechende Rechere und Selbststudium die erforderlichen Implementierungstechniken anzueignen. Dafür machen sie aber im Gegenzug zum Teil erhebliche Abstriche bei der Modellierung, welche jedoch das eigentliche „offizielle“ Lernziel der ersten Software-Engineering-Lehrveranstaltung gewesen wäre. Außerdem wird Software Engineering 2 von diesem Personenkreis dann gerne als langweilig empfunden, weil die benötigten Inhalte ja schon zum großen Teil in Software Engineering 1 autodidaktisch erschlossen wurden.

Software Engineering Lehre nach dem Wasserfallmodell

Über ein ganzes Semester hinweg betrachtet sind viele Software Engineering Lehrveranstaltungen so aufgebaut, dass sie zunächst einen sehr groben Überblick über den Stoff des gesamten Kurses vermitteln. Anschließend werden Woche für Woche die zu behandelnden Themen abgearbeitet, in der Regel streng sequenziell hintereinander. Jedes Thema wird dabei umfassend und erschöpfend behandelt, bevor zum nächsten Thema vorangeschritten wird.

Beispielsweise lernen Studierende im Software Engineering auf diese Weise also zunächst alle Facetten der Use-Case-Modellierung kennen. Wenn diese abgeschlossen sind folgen Ablaufbeschreibungen mittels Aktivitätsdiagrammen. Danach werden Klassendiagramme durchgenommen, gefolgt von Sequenzdiagrammen, und so weiter (Abbildung 1). Wenn die ganzen Modellierungsthemen abgehandelt sind folgt im nächsten großen Themenblock dann die Softwarearchitektur.

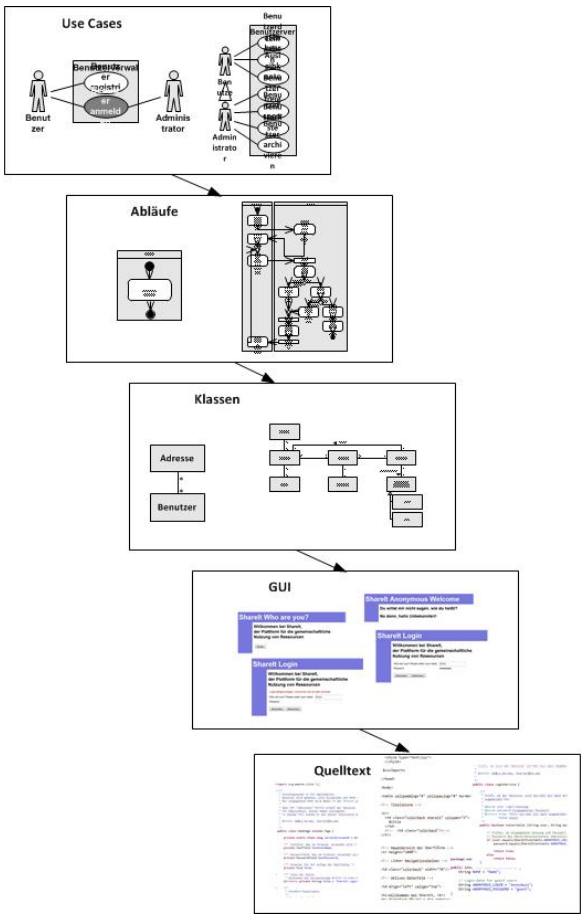

Abbildung 1: Lehre nach dem Wasserfallmodell

Der Vorteil dieses Ansatzes liegt darin, dass die Veranstaltung klar gegliedert ist. Insbesondere wird jedes Thema umfassend und abschließend behandelt. Des Weiteren sind die Themen klar voneinander abgegrenzt. Sorgfältige Mitschriften aus so aufgebauten Veranstaltungen können sich gut als Refernz für die Prüfungsvorbereitung bzw. als Nachschlagewerk eignen, weil alle Informationen, die zu einem Themenbereich gehören, im zugehörigen Kapitel zu finden sind.

Letzten Endes ist dieser didaktische Ansatz analog zum Wasserfallmodell im Software Engineering, mit vergleichbaren Vor- und Nachteilen. So erkauft man sich mit dem Vorteil der klaren Struktur den (gewichtigen) Nachteil, dass Querbeziehungen zwischen Lerninhalten aus frühen und aus späten Lehr-/Lernphasen erst sehr spät aufgedeckt werden. Des Weiteren bleiben bei der übenden bzw. praktischen Anwendung des Gelernten die Auswirkungen von Entscheidungen aus frühen Phasen relativ lange unklar für die Studierenden, da der Round Trip über den gesamten Software Life Cycle sich oft über mehrere Wochen oder Monate hinzieht und die ausführbaren Ergebnisse erst entsprechend spät vorliegen.

Iterativ-inkrementelle Vermittlung von Software Engineering Wissen

Aus diesem Dilemma heraus entstand die Idee, den Grundgedanken der heute in der Praxis vorherrschenden (Rupp u. Joppich, 2009) iterativen, inkrementellen Entwicklung auch auf den Lernprozess für angehende Softwareingenieure zu übertragen.

Zur Umsetzung dieses didaktischen Ansatzes vermittelt der seminaristische Unterricht genau diejenigen theoretischen Grundkenntnisse, die für das jeweilge Inkrement erforderlich sind. Jede Lehr-/Lerneinheit berührt somit die verschiedenen Kerndisziplinen der Entwicklung (mit ggf. unterschiedlicher Gewichtung), also mehrere Entwicklungsschritte, mehrere UML-Diagrammtypen, mehrere Implementierungskonzepte und ggf. mehrere Werkzeuge. Dafür wird in einem Inkrement jeder dieser Lernbereiche jeweils nur um ein kleines Stückchen Information und Wissen erweitert.

Parallel dazu wird im Praktikum ein kleines System iterativ und inkrementell modelliert und entwickelt. Jedes Inkrement deckt eine komplette Iteration durch den Software Life Cycle ab. Die initialen Inkremente sind zunächst sehr klein und einfach und mit viel detaillierter Anleitung hinterfüttert. Je weiter das Projekt fortschreitet, umso anspruchsvoller werden die Inkremente und die dafür benötigten Werkzeuge und Technologien. Essenziell dabei ist, dass am Ende jeder Iteration ein lauffähiges System entsteht, anhand dessen sich die Zusammenhänge zwischen den einzelnen Disziplinen, verwendeten Technologien und angewendeten methodischen Vorgehensweisen unmittelbar erkennen lassen.

Damit die Studierenden ein Verständnis dafür entwickeln, dass sie im Rahmen des Lehrkonzeptes lediglich eines von mehreren möglichen Vorgehensmodellen durchführen wurde entgegen der inhaltlichen Aufteilung in der Modulbeschreibung die Themen Pro-

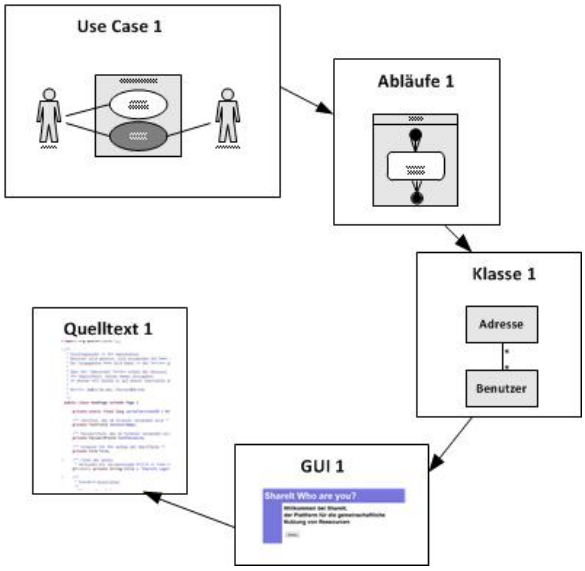

Abbildung 2: Einfaches Inkrement 1

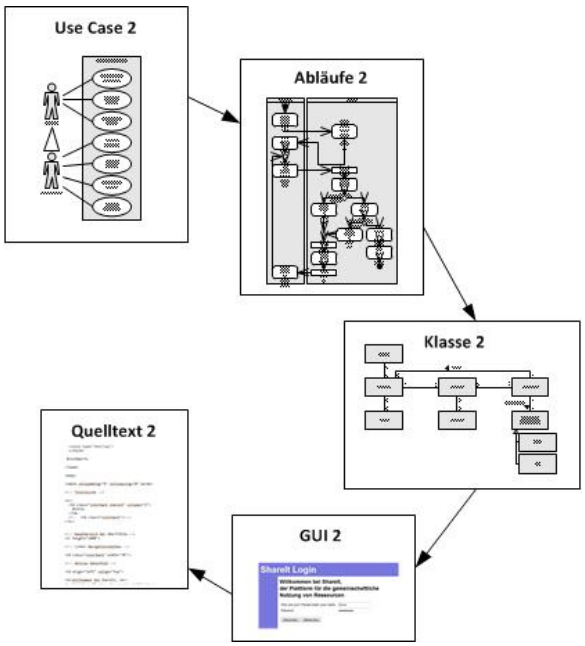

Abbildung 3: Komplexeres Inkrement 2

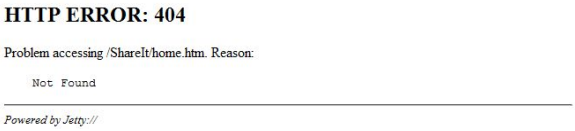

Abbildung 4: JettyServer mit Standardfehlerseite

zessmodelle und agile Vorgehensmodelle aus dem 4. Semester in das 3. Semester vorgezogen. Im Gegenzug wurden die Themen der Aufwandsschätzung und des Projektmanagements vom 3. in das 4. Semester verlagert.

Struktur des zentralen Beispiels

Kern des Lehrkonzeptes ist ein durchgängiges Beispiel, das in Grundzügen vorentwickelt ist und von den Studierenden im Rahmen des Praktikums sukzessive erweitert wird. Eine Urversion davon wurde bereits in (Thurner u. Böttcher, 2010) vorgestellt.

Die folgende Aufzählung skizziert knapp die Funktionalität, die sukzessive in den einzelnen Inkrementen umgesetzt wird. Darauf aufbauend verdeutlicht Tabelle 1, welche Techniken und Wissensbereiche in den einzelnen Inkrementen jeweils zum Einsatz kommen.

1. Der Administrator startet einen Jetty-Server als leichtgewichtigen Application Server. Anschließend greift der Normalbenutzer über den Browser auf den lokalen Port zu, unter dem der Jetty-Server bereit steht. Anhand der angezeigten Standard-Fehlerseite ist erkennbar, dass der Application Server im Hintergrund läuft (siehe Abbildung 4).

2. In der zweiten Aufbaustufe wird die Anfrage an einen Handler weitergeleitet, die als Antwort den minimalistisch formatierten statischen Text „Hello World“ erzeugt.

3. Das nächste Inkrement führt das Zusammenspiel von Java-Klasse und parametrisierter Webseite ein. Dabei ist das titel-Attribut der Java-Klasse an die $titel-Variable der Webseite gebunden. Des Weiteren nutzt die Webseite CSS und HTML zur Formatierung der Inhalte (siehe Abbildung 5).

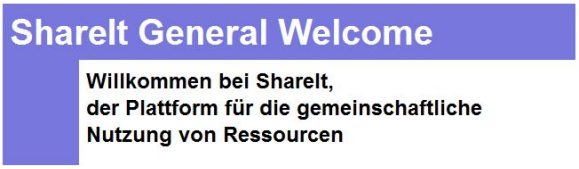

Abbildung 5: Startseite mit variablem Titel

4. Ein erster Button ist das zentrale neue Element der nächsten Ausbaustufe (Abbildung 6). Für dessen Umsetzung wird die Verwendung eines Formulars und eine Listener-Methode eingeführt sowie auf eine Antwortseite weitergeleitet (Abbildung 7).

   Die Modellebene wird ergänzt um ein einfaches Sequenzdiagramm, welches das Zusammenspiel der beteiligten Klassen veranschaulicht. Des Weiteren wird das Aktivitätsdiagramm erweitert zu einem noch recht einfachen Ablauf, bei dem Benutzer und System in mehreren Schritten miteinander interagieren.

5. Darauf aufbauend umfasst das nächste Inkrement ein Textfeld, in das der Benutzer seinen Namen eingeben kann (Abbildung 8). Nach Absenden

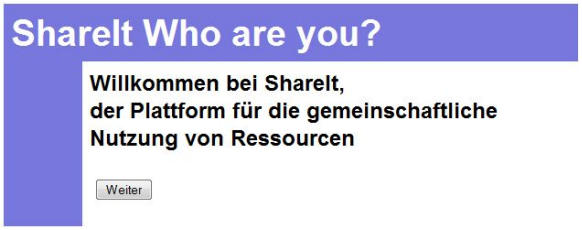

Abbildung 6: Startseite mit Button

Abbildung 7: Statische Antwortseite

der Anfrage über den Button erscheint eine personalisierte Begrüßungsseite (Abbildung 9). Für deren Umsetzung wurde der eingegebene Datenwert in der Session abgelegt und zum Generieren der Antwortseite wieder ausgelesen.

6. Die nächste Ausbaustufe bietet dem Benutzer auf der Startseite zwei Buttons an und erlaubt so eine Auswahlmöglichkeit (Abbildung 10). Entsprechend wird das Aktivitätsdiagramm um die Konzepte der Fallunterscheidung und des Ereignisses erweitert. Auch das Sequenzdiagramm wird um alternative Blöcke ergänzt.

7. Das folgende Inkrement realisiert eine Login-Möglichkeit über eine Gastkennung. Dazu wird das System um eine neue Komponente erweitert, die die Business Services kapselt. Zur Qualitätssicherung der Business Services wird deren Funktionalität mit Hilfe von Unit-Tests abgesichert. Die GUI erhält als neues Element ein Passwort-Feld. Zu den bisher bekannten Notationselementen in Aktivitäts- und Sequenzdiagrammen werden nun Rückkopplungen über join-Knoten bzw. Wiederholungsblöcke hinzugefügt.

8. Weitere Ausbaustufen fokussieren Schritt für Schritt die Realisierung von Entity-Klassen, deren Persistierung in einer Datenbank sowie die Umsetzung von Menüs zur Benutzerführung. Parallel zu den Konzepten der technischen Umsetzung werden auch hier die benötigten Modellierungsnotationen sukzessive weiter ausgebaut.

Erste Erfahrungen

Der hier vorgestellte Lehr-/Lernansatz wird in diesem Semester erstmals auf diese Weise an der Hochschule München erprobt.

Die zentrale Herausforderung in der Vorbereitungsphase bestand darin, in jedem Schritt jeweils nicht zuviel Lernstoff auf einmal anzugehen, sondern statt

Abbildung 8: Startseite mit Texteingabefeld

Abbildung 9: Personalisierte Begrüßungsseite

dessen möglichst simpel anzufangen und die nachfolgenden einzelnen Inkremente angemessen klein zu halten. Dazu wurde das ursprüngliche Beispiel (Thurner u. Böttcher, 2010) solange sukzessive immer weiter abgespeckt und vereinfacht, bis es nur noch die wesentlichen zu zeigenden Inhalte, Techniken und Zusammenhänge umfasst.

Die daraus resultierende minimalisierte Version von Spezifikation und System war für den Einstieg jedoch immer noch viel zu komplex. Daher wurde in einem zweiten Schritt, noch einmal ganz von vorne mit einem leeren Projekt anfangend, eine Folge von Spezifikationen und Systemen erstellt, die von der Komplexität und vom Funktionsumfang her bei null anfängt und in winzigen Einzelschritten sukzessive aufgebaut wird. Am Ende dieser Folge steht die minimalisierte Version, auf die das urspüngliche umfangreiche Beispiel zuvor eingedampft worden war.

Nach den bisher gewonnenen ersten Erfahrungen kommen die Studierenden gut mit dem iterativ-inkrementellen Lernansatz zurecht. Insbesondere werden die Zusammenhänge zwischen den einzelnen Disziplinen und Techniken schneller und deutlicher wahrgenommen als beim eher sequenziell orientierten Aufbau früherer Veranstaltungen. Des Weiteren erscheinen durch die relativ kleinen Schritte von Inkrement zu Inkrement die Studierenden weniger überfordert, als dies beispielsweise beim in der Vergangenheit angewendeten Lehransatz mit Gruppenpuzzle und Lernen durch Lehren der Fall war.

Dadurch, dass beim iterativen Vorgehen bereits behandelte Inhalte und Wissensbereiche immer wieder aufgegriffen und inkrementell weiter ausgebaut werden, werden außerdem die Kerninhalte regelmäßig wiederholt und scheinen sich so besser einzuprägen.

Nachteilig beim iterativ-inkrementellen Lehr-/Lernansatz ist, dass bei dieser Vorgehensweise die Informationen zu den einzelnen behandelten Themenbereichen (wie z. B. einzelne Diagrammtypen der UML) quer über die Lehrmaterialien verteilt sind, da jedes

Wissensbereich	Inkr. 1	Inkr. 2	Inkr. 3	Inkr. 4	Inkr. 5	Inkr. 6	Inkr. 7
Use Case Diagramm	Akteur, Use Case, Assoziation	Systemgrenze, Business / System Use Case					
Use Case Beschreibung		textuelle Kurzbeschreibung, Vorbedingung, Ergebnis	Normalablauf, Fehlerfall			Alternativer Ablauf	
Aktivitätsdiagramm	Start, Stop, Aktion, Kontrollfluss	Schwimmbahn		Folge von Aktionen		Fallunterscheidung, Ereignis	Wiederholung
Komponentendiagramm			Komponente, Beziehung				Schnittstelle
Sequenzdiagramm				Kommunikationspartner, Lebenslinie, Nachricht, Aktionssequenz	Nachricht mit Methodenaufruf, Antwort	Alternative Blöcke	Schleifen-Block
Webtechniken	Webapplikation	HTML, HTTP, Request, Response, Bedeutung Servlet	CSS, Trennung von Darstellung und Logik	Weiterleiten auf Antwortseite	Konzept der Session, Datentransfer über Session		
Architektur	Bedeutung Application Server	Handler	Schichtenarchitektur	Listener-Methode			Business Service
GUI-Framework (Apache Click)			Startseite, Zusammenspiel Klasse und Webseite, @Bindable Attribut, $Variable	Rendering, onInit(), Form, Control, Button	Textfeld		Passwort-Feld, Fehlermeldung
Unit-Test							Unit-Tests für den Business Service

Tabelle 1: Wissenselemente, die in den ersten Inkrementen der Systemfolge behandelt werden

Abbildung 10: Auswahlmöglichkeit durch zwei Buttons

Thema mehrfach und in zunehmenden Detaillierungsstufen aufgegriffen wird.

Abhilfe lässt sich hier schaffen durch Bereitstellung entsprechender Begleitliteratur, sofern diese themenweise aufgebaut ist. Auch die Erstellung eines ergänzenden Skriptes wäre natürlich denkbar, war jedoch aus Zeitgründen vor dieser ersten Umsetzungsphase noch nicht möglich. Statt dessen werden die Studierenden dazu angeleitet, sich als Nachschlagewerk eigene Zusammenfassungen zu den einzelnen Themengebieten zu erstellen und diese semesterbegleitend kontinuierlich auf- und auszubauen. Diese dürfen (auf 5 beidseitig beschriebene A4-Blätter beschränkt) dann als Hilfsmittel in die Prüfung mitgenommen werden.

Die obigen Aussagen spiegeln ausschließlich den Eindruck wider, den die Dozentin in den Lehrveranstaltungen und aus Gesprächen mit den Studierenden gewonnen hat. Prüfungsergebnisse, die als Kennzahl für die Bewertung des Lernerfolgs herangezogen und mit den Ergebnissen anderer Ansätze verglichen werden könnten, liegen zum aktuellen Zeitpunkt jedoch noch nicht vor.

In vergangenen Durchführungen der Lehveranstaltung Software Engineering 1 lag der Fokus überwiegend auf der Modellierung und den frühen Phasen, wie in der Modulbeschreibung vorgegeben. Durch das iterativ-inkrementelle Lehrkonzept sind Aspekte der Implementierung stärker in den Vordergrund gerückt. Dies wird sich in der Gestaltung der Prüfung entsprechend widerspiegeln, um dadurch die von den Studierenden erworbenen Kompetenzen im Bereich des Zusammenspiels von Modellierung und Implementierung zu bewerten.

Bei der initialen Umsetzung des Lehrkonzeptes erwies sich die Granularität der einzelnen Inkremente immer wieder als zentraler Knackpunkt. War die Menge an neuen Konzepten klein genug und deren Anbindung an bereits erarbeitetes Vorwissen ausreichend, so erforderte die anschließende Betreuung der Studierenden in den Praktika in etwa vergleichbaren Aufwand wie in den Vorjahren. Erwies sich die Granularität bzw. Komplexität des Inkrementes dagegen als zu hoch, schnellte der Betreuungsaufwand extrem in die Höhe.

Des Weiteren war zu beobachten, dass die Motivation der implementierungsstarken Studierenden in der Veranstaltung durch die enge Verzahnung zwischen Modellierung und Umsetzung erheblich gestiegen ist, ebenso wie deren Verständnis für die Notwendigkeit und die Aussagekraft der "bunten Bildchen".

Die Teilkohorte der implementierungsschwächeren Studierenden hatte dagegen teilweise erhebliche Probleme mit der Bewältigung der Praktikumsaufgaben. Durch die Notwendigkeit, die erstellten Modelle unmittelbar in ein lauffähiges System umzusetzen, wurden Lücken aus den implementierungsnahen Grundlagenfächern erneut evident. Darüber hinaus erzwang die drohende Umsetzung der Modelle bereits eine hohe Sorgfalt und Präzision in den Diagrammen, welche bei einer reinen Modellierungsaufgabe nicht im gleichen Maße zweifelsfrei offensichtlich geworden wäre. Es war daher aufgrund dieser engen Verzahnung weniger leicht möglich, sich durch die Aufgaben “irgendwie durchzuwursteln", was bei einigen Studierenden zu einer deutlich realistischeren Einschätzung des eigenen Leistungsstandes geführt hat. Das war teilweise sehr heilsam, hat aber bei einigen auch zu nicht unerheblichem Frust geführt.

Zusammenfassung und Ausblick

Der hier vorgestellte iterativ-inkrementelle Lehr-/Lernansatz für Wissensbereiche und Kompetenzen des Software Engineerings stellt eine Möglichkeit dar, die umfangreichen und stark miteinander verzahnten Themengebiete so aufzuschlüsseln, dass die einzelnen Lehr-/Lerneinheiten einerseits nicht überfrachtet sind und andererseits die Zusammenhänge zwischen den Wissensbereichen frühzeitig deutlich erkennbar und für die Studierenden auch selbst praktisch nachvollziehbar werden.

Erste Erfahrungen aus der Umsetzung des Ansatzes verlaufen bisher vielversprechend. Aktuell werden der Lehransatz und die benötigten Materialien kontinuierlich erweitert und ausgebaut. Eine Messung des Lernerfolges über Prüfungsleistungen steht derzeit noch aus.

Interessant ist des Weiteren auch die Frage, ob der systemimmanente hohe Wiederholunganteil des iterativ-inkrementellen Lehr-/Lernansatzes auch die Nachhaltigkeit des Gelernten positiv beeinflusst. Um hier eine Aussage treffen zu können ist jedoch eine Langzeitstudie mit entsprechenden Vergleichsgruppen erforderlich.

Literatur

[Abran u. a. 2005] ABRAN, A. ; MOORE, J. ; DUPUIS, R. ; TRIPP, L.: *Guide to the Software Engineering Body of Knowledge 2004 Version SWEBOK*. IEEE Computer Society Press, 2005

[IEEE 2012] IEEE, Computer-Society: *Software Engineering Body of Knowledge, vorläufiger Stand der Version 3*. www.computer.org/portal/web/swebok, abgerufen am 28.10.2012, 2012

[Lehner 2009] LEHNER, M.: *Viel Stoff – wenig Zeit*. Haupt Verlag, Stuttgart, 2009

[Rupp u. Joppich 2009] RUPP, C. ; JOPPICH, R.: *Dokumentenberge oder Bierdeckel – Requirements Engineering in Zeiten der Agilität*. heise Developer, http://www.heise.de/developer/artikel/Requirements-Engineering-in-Zeiten-der-Agilitaet-804971.html, abgerufen am 28.10.2012, 2009

[Thurner u. Böttcher 2010] THURNER, V. ; BÖTTCHER, A.: Beispielorientiertes Lernen und Lehren im Software Engineering. In: *ESE 2010: Embedded Software Engineering Kongress*, 2010, S. 591–595