

# Kompetenzorientierte Lehre im Software Engineering

Axel Böttcher und Veronika Thurner, Hochschule München  
ab | thurner@cs.hm.edu

Gerhard Müller, TNG Technology Consulting GmbH  
gerhard.mueller@tngtech.com

## Zusammenfassung

Software Engineering ist eine komplexe Aufgabe, die von den Beteiligten ausgeprägte Kompetenzen in vielen verschiedenen Bereichen fordert. Die Vermittlung dieser Kompetenzen über das reine Fachwissen hinweg ist eine zentrale Aufgabe, die eine Software-Engineering-Ausbildung leisten muss, um Absolventen für den Einsatz in der Praxis zu qualifizieren. Ausgehend von den zu vermittelnden Kompetenzen zeigen wir, wie wir in unserer Lehrpraxis mit Hilfe eines möglichst einfach gehaltenen, aber so komplex wie nötig gestalteten durchgängigen Beispiels sowie durch innovative Lehrmethoden praxisnah eine fundierte Einführung in die komplexen Tätigkeiten des Software Engineerings vermitteln.

## Motivation

Anforderungen an Softwaresysteme werden heute zunehmend umfangreicher und komplexer. Analog dazu entwickeln sich auch die Technologien und Werkzeuge für Software Engineering stetig weiter und werden immer mächtiger – und damit selbst komplexer. Da sich diese Technologien und Werkzeuge laufend weiter entwickeln, ist die Halbwertszeit der dazu erworbenen Kompetenzen entsprechend kurz.

Aufgrund der Menge und Komplexität des erforderlichen Wissens und wegen der Größe der zu erstellenden Systeme wird Software in der heutigen Praxis nahezu immer im Team entwickelt. Neben den rein technisch-fachlichen Kompetenzen werden damit in zunehmendem Maße auch Fähigkeiten wichtig, die sich mit der Positionierung einer Einzelperson im Team sowie mit der Integration und Interaktion aller Beteiligten untereinander befassen.

Eine Software-Engineering-Ausbildung, die nicht nur theoretisch fundiert, sondern auch praxisnah und berufsbefähigend sein soll, muss diesen Entwicklungen Rechnung tragen. Hier stößt die Lehre sehr schnell auf das heute bereits klassische Dilemma, dass die verfügbare Lernzeit in der initialen Ausbildungsphase eines Menschenlebens seit Jahren in etwa konstant bleibt, die Menge des verfügbaren und als wichtig empfundenen Wissens aber exponentiell an-

steigt (Spitzer, 2006). Da dieser Entwicklung alleine mit erhöhter Packungsdichte in der Wissensvermittlung nicht mehr beizukommen ist, ist zwangsweise Selektion erforderlich. Eine „gute“ Selektion wird somit zu einer der wesentlichen Herausforderungen bei der Konzeption der Software-Engineering-Ausbildung.

Im Folgenden definieren wir zunächst kompetenzorientierte Lernziele für die Software-Engineering-Ausbildung. Diese basieren im Wesentlichen auf drei Quellen:

- klassischer Software-Engineering-Literatur,
- einer umfangreichen Befragung von Absolventen unserer Fakultät aus den letzten fünf Jahrgängen sowie
- Aussagen von Spezialisten aus der beruflichen Praxis über die Eigenschaften und Fähigkeiten, die sie von potenziellen neuen Mitarbeitern/innen erwarten.

Im Anschluss daran beschreiben wir ein Beispielprojekt und die zugehörigen Lehrkonzepte, die wir in den Bachelor-Studiengängen Informatik sowie Wirtschaftsinformatik einsetzen. Abschließend setzen wir die mit unserem Lehrkonzept gemachten Erfahrungen in Beziehung zu den zuvor definierten Kompetenzzielen.

## Definition von Lernzielen

Grundlage einer adäquaten Selektion von Ausbildungsinhalten und -methoden ist die Definition der zu erreichenden Lernziele. In der Regel stellt die Definition dieser Lernziele bereits selbst eine Auswahl dar, da in der meist begrenzten verfügbaren Zeit realistischer Weise nicht immer alle eigentlich gewünschten Ziele erreicht werden können (Lehner, 2009).

Ausbildung und Lernziele müssen sich dabei zum einen an Kompetenzen auf unterschiedlichen Wissensgebieten orientieren. Zum anderen sollen sie die Studierenden auf die Übernahme verschiedener Rollen vorbereiten. Analog zu Schott und Ghanbari (Schott u. Ghanbari, 2009) betrachten wir Kompetenzen als diejenigen Eigenschaften und Fähigkeiten, die man besitzen muss, um eine bestimmte Menge von Aufgaben sinnvoll ausführen zu können.

Softwarekompetenzen (SWEBOK)	Inf	WI
Requirements	++	++
Design	++	++
Construction	++	+
Testing	++	+
Maintenance	-	-
Configuration Management	o	o
Engineering Management	o	+
Engineering Process	+	++
Engineering Tools and Methods	+	+
Quality	+	+

Tabelle 1: Unsere Prioritäten für die Vermittlung der Kompetenzen (-, -, o, +, ++) in den Bachelor-Studiengängen Informatik (Inf) und Wirtschaftsinformatik (WI)

Als Grundlage für die Definition von zu vermittelnden Kompetenzen und daraus abzuleitenden Lernzielen, empfiehlt sich zunächst der „Software Engineering Body of Knowledge“ (Abran u. a., 2005). Die dort genannten Kompetenzfelder haben wir für die Durchführung der Lehre in den Bachelor-Studiengängen Informatik und Wirtschaftsinformatik priorisiert und in Tabelle 1 zusammengefasst. Der Themenbereich „Software Maintenance“ wird aktuell nur am Rande adressiert. Das liegt unter anderem daran, dass Software Maintenance zum einen sehr technologiespezifisch ist und zum anderen stark auf anderen Kompetenzen aufbaut. Diese müssen somit zuerst vermittelt werden, bevor diese Thematik auf technisch anspruchsvollem Niveau behandelt werden kann.

Als ergänzende Informationsquelle haben wir eine Umfrage herangezogen, die unter unseren Absolventen der Informatik und Wirtschaftsinformatik der letzten Jahre durchgeführt wurde – also unter Personen, die sich bereits mit den bei uns erworbenen Fähigkeiten in der Industrie bewähren müssen. Darüber hinaus befragten wir erfahrene Praktiker aus der Industrie, welche konkreten Anforderungen die Praxis an unsere Absolventen stellt. Die auf diese Weise identifizierten Kompetenzbedarfe gliedern wir im Folgenden nach den vier Bereichen Sach-, Methoden-, Selbst- und Sozialkompetenz (vergleiche (Schaeper u. Briedis, 2004)). Dabei ordnen wir Kompetenzen, die mehr als einen dieser Bereiche bedienen, demjenigen Kompetenzbereich zu, zu dem sie nach unserer Einschätzung den stärksten Beitrag leisten.

## Sachkompetenz

Der Bereich der Sachkompetenz fokussiert das erworbene Fachwissen und dessen zielgerichtete Anwendung. Hierzu zählen die in Tabelle 1 aufgelisteten Kompetenzbereiche aus dem SWEBOK. Diese wurden aus der Praxis durch die folgenden Kompetenzforderungen ergänzt.

- *Modellierung und Entwurf*  
Um eine „ordentliche“, d.h. professionelle und nachhaltige Softwareentwicklung rein technisch-handwerklich überhaupt zu ermöglichen müssen Software-Ingenieure/innen modellieren und designen können. Dazu gehören unter anderem auch das Verständnis für bzw. der Einsatz von Design Patterns, SOLID-Prinzipien und ggf. Domain Driven Development (Evans, 2003).
- *Beschreibung von Architekturen*  
Software-Ingenieure/innen müssen in der Lage sein, technische und fachliche Architekturen zu konzipieren und so darzustellen, dass sie als klar verständliche Grundlage für die Kommunikation aller Projektbeteiligten dienen können.
- *Einarbeitung in große, fremde Projekte*  
Softwaresysteme entstehen heute häufig im Kontext bereits bestehender Systemlandschaften. Eine wichtige Fähigkeit von Software-Ingenieuren/innen ist somit die Einarbeitung in umfangreiche fremde Projekte. Typische Aufgabenstellung ist das Bug-Fixing in fremdem Code, aber auch das Einbauen neuer Features in ein bestehendes System.
- *Gesamtsicht auf Zusammenspiel von Frameworks*  
Moderne Softwaresysteme sind gekennzeichnet durch das Zusammenwirken vieler Bibliotheken und Frameworks. Jeder dieser Bausteine ist für sich genommen nicht schwierig zu verstehen. Die Integration dieser Frameworks in einem Produkt erhöht die Komplexität jedoch sprunghaft. Entsprechend müssen diese Bausteine in der Lehre nicht nur einzeln, sondern auch in ihrer Gesamtsicht vermittelt werden.
- *Test Driven Development*  
Test Driven Development ist in der heutigen Praxis eine Basistechnik der professionellen, nachhaltigen Softwareentwicklung. Damit einher gehen das Erlernen eines sauberen Designs, Continuous Integration, Refactoring und Techniken wie das Pair Programming.
- *Verständnis für Lebensdauer von Code*  
Die Praxis zeigt, dass der Quelltext geschäftsrelevanter Softwaresysteme oft über Jahrzehnte im Einsatz bleibt. Entsprechend müssen Software-Ingenieure/innen ein Verständnis für die Lebensdauer von Code entwickeln die Wichtigkeit von Informationen erkennen, die in „Clean Code“ beschrieben sind (Martin, 2008). Darüber hinaus müssen Sie in der Lage sein, selbst sauberen Code zu schreiben. Ebenfalls erforderlich ist ein Verständnis für die Kosten von Zeit.
- *Größenordnungen verstehen*  
In der Praxis bestimmen nicht-funktionale Requirements oft zu ca. 80% die Systemarchitek-

tur, während die funktionalen Anforderungen im Vergleich dazu eine eher untergeordnete Rolle spielen. Um auf systematische Weise adäquat performante Systemarchitekturen konzipieren zu können müssen Software-Ingenieure/innen ein Gefühl für Größenordnungen entwickeln, beispielsweise bzgl. algorithmischer Komplexität, Anzahl der Transaktionen pro Sekunde, Speicherbedarf pro Session, IO-Geschwindigkeit, CPU-Geschwindigkeit, Bandbreite, Ressourcenbedarf pro Benutzer oder in Summe, ... Darüber hinaus ist ein Verständnis für die Auswirkungen dieser Werte erforderlich sowie die Fähigkeit, ggf. passende Konsequenzen daraus herzuleiten. Ohne dieses Verständnis werden die erarbeiteten Lösungen nicht auf angemessene Weise skalierbar.

- *Aufwandsschätzung*  
Aufwandsschätzungen bilden nicht nur die Grundlage für viele planerischen und organisatorischen Tätigkeiten im Software Engineering, sondern beispielsweise auch für die Entscheidung zwischen verschiedenen Realisierungsalternativen. Um verlässliche Aussagen zu Machbarkeit, Zeit- und Ressourcenbedarf treffen zu können müssen Software-Ingenieure/innen Aufwände auf realistisch treffende Weise schätzen können.
- *Vorgehensmodelle*  
Der in der Theorie gelegentlich verfolgte Grundgedanke eines universell einsetzbaren Vorgehensmodells („One size fits all“) funktioniert in der Praxis in der Regel nicht. Entsprechend müssen Software-Ingenieure/innen verschiedene Vorgehensmodelle wie z.B. Scrum, XP, Crystal-Familie, RUP und V-Modell-XT kennen und deren Vor- bzw. Nachteile und Einsatzgebiete einschätzen können. Insbesondere sind Kompetenzen sowohl zu klassisch ingenieurwissenschaftlichen als auch zu aktuellen agilen Vorgehensmodellen von Nöten.
- *Beherrschung grundlegender Werkzeuge*  
Damit alle Beteiligten auf technischer Ebene reibungsfrei zusammen arbeiten können gehört die Beherrschung grundlegender Werkzeuge zum elementaren Handwerkszeug von Software-Ingenieuren/innen, wie beispielsweise die verwendete IDE oder die Versionsverwaltung incl. Kenntnissen zu geeigneten Branching-Strategien.

### Methodenkompetenz

Im Bereich der Methodenkompetenz sind grundlegende Arbeitstechniken zusammengefasst, sowie Fähigkeiten und Verfahren, die ein effektives, ergebnisorientiertes Arbeiten ermöglichen.

- *Denken in Systemen*  
Wer nicht in übergeordneten Strukturen denkt, beschränkt sich auf lokale Optimierungen und tendiert dazu, „echte“ Probleme nicht anzugehen. Strukturprobleme anzugehen erfordert auch Mut.

- *„Faulheit“, DRY-Prinzip*  
Das Prinzip des „Don't Repeat Yourself“ bedeutet, im Normalfall keine eigenen Frameworks bauen wollen, Erkennen der Notwendigkeit Automatisierung auf allen Ebenen in möglichst großem Umfang zu erreichen. Software lebt von der Wiederverwendung bewährter Konzepte und Lösungen. Die so genannten „Not Invented Here“-Lösungen bedeuten eine Verschwendung von Ressourcen.
- *Vernünftiges Maß an Pragmatismus*  
In der beruflichen Praxis müssen ständig Entscheidungen getroffen werden, ob etwa die Menge der Tests ausreichend ist, ob ein noch abstrakteres Framework einzusetzen ist, oder ob an einer Stelle noch mehr Optimierungsaufwand getrieben werden soll etc. Um in solchen für ein Projekt kritischen Fragen vernünftig abwägen zu können, ist viel Erfahrung erforderlich.
- *Prägnantes Dokumentieren von Anforderungen*  
Die in Projekten auftretenden Probleme sind oftmals weniger technischer Natur als vielmehr organisatorischer Art. Software-Ingenieure/innen müssen oft zuerst herausfinden, welches Problem der Kunde eigentlich gelöst haben will. Hier ist es insbesondere wichtig, Requirements kurz und prägnant aufschreiben zu können.
- *Ausdrucks- und Schreibfähigkeit*  
Software-Ingenieure/innen müssen in der Lage sein, aussagekräftige, klar verständliche Dokumente und Dokumentation zu schreiben. „Ordentliche“ Dokumente sind in der Praxis eine zentrale Voraussetzung dafür, dass Lösungen bzw. Lösungs-ideen von den Beteiligten verstanden und von den Nutzern akzeptiert und eingesetzt werden und dass Lösungen weiterentwickelbar sind, ggf auch über einen längeren Zeitraum und mehrere Entwicklungsgenerationen hinweg.

### Selbstkompetenz

Zum Bereich der Selbstkompetenz zählen Fähigkeiten, die eigene Situation wahrzunehmen, eigene Bedürfnisse zu erkennen und zu artikulieren, eigenverantwortlich zu handeln und über all diese Punkte selbstkritisch und konstruktiv zu reflektieren.

- *Reflexions- und Kritikfähigkeit*  
Reflexions- und Kritikfähigkeit sind Grundvoraussetzungen für eine erfolgreiche Berufstätigkeit, nicht nur im Umfeld des Software Engineerings. Ziel der Reflektion ist es, sich selbst und die (Projekt-)Umgebung immer wieder zu hinterfragen, um so sicherzustellen, dass „das Richtige“ getan wird. Kritikfähigkeit ermöglicht es, das ggf. kritische Feedback Anderer nicht als persönlichen Angriff zu werten, sondern als Verbesserungspotenzial anzuerkennen und daraus zu lernen. Auch

das Geben von konstruktiv verwertbarem Feedback ist in diesem Kontext eine wesentliche Schlüsselkompetenz.

- *Dinge „richtig“ machen*  
Die Informatik allgemein, und damit auch das Software Engineering, erfordern nicht nur eine präzise Denkweise und ein hohes Maß an Wissen, sondern auch eine ständige Aktualisierung dieses Wissens, um mit den ständig neuen Entwicklungen Schritt halten zu können. Der Wunsch, Dinge „richtig“ zu machen, gepaart mit dem Streben nach lebenslangem Lernen und Verbessern, sollte in Software-Ingenieuren/innen intrinsisch motiviert sein. *Ohne* diese Eigenschaft werden die vielen Innovationen und die steigende Komplexität eher als Belastung empfunden und nicht als Anreiz. In einer solchen Konstellation *kann* die Informatik mit ihren vielen Änderungen auf Dauer keinen Spaß machen – und dann werden auch keine vernünftigen Ergebnisse erzielt.
- *Ethisches Handeln*  
Softwaresysteme durchziehen heute nahezu alle Bereiche des beruflichen und privaten Lebens. Damit beeinflusst die Informatik in hohem Maße, wie andere Menschen leben und arbeiten. Software-Ingenieure/innen gestalten somit mehr oder weniger direkt die Zukunft. Dies bringt nicht nur eine kaum absehbare Vielzahl an (Geschäfts-)Möglichkeiten mit sich, sondern auch ein sehr hohes Maß an Verantwortung für die Rolle der Informatik in der Welt. Eine ganzheitliche Software-Engineering-Ausbildung sollte in angehenden Software-Ingenieuren/innen zumindest das Bewusstsein für diese Verantwortung wecken, sowie ein gewisses Maß an kritischem Weitblick aufbauen und fördern.

### Sozialkompetenz (Soft Skills)

Der Bereich der Sozialkompetenz fokussiert schließlich die Fähigkeit, die Bedürfnisse und Interessen anderer Menschen wahrzunehmen, sich mit diesen auseinander zu setzen und konstruktiv und erfolgreich mit anderen Menschen zusammenzuarbeiten.

- *Verständnis für Andere*  
Software entsteht nicht in einem luftleeren Raum, sondern durch die Zusammenarbeit verschiedenster Gruppen, wie beispielsweise Management, Benutzer, Fachexperten, QA, Operations und vielen anderen. Schon die Zielfindung in einem Entwicklungsprojekt erfordert intensive Kommunikation zwischen den Beteiligten sowie eine offene Wahrnehmung der Belange und Interessen aller Beteiligten.
- *Teamfähigkeit*  
In der Praxis ist Software heutzutage fast immer das Produkt eines Teams. Entsprechend kommt

der Fähigkeit, sinnvoll, effektiv und gerne (!) in einem Team zusammenzuarbeiten eine fundamentale Bedeutung zu: Wer das nicht kann, kann nichts von dem, was er/sie kann, sinnvoll in komplexeren Umgebungen einbringen. Zu den zentralen Teilfähigkeiten gehören unter anderem eher pragmatische Punkte wie ein freundlicher Umgangston, nett sein und zuhören können. Gerade in kritischen Situationen gewinnt darüber hinaus theoretisch fundiertes Wissen über Team-Regeln an Bedeutung, wie beispielsweise das Tucker-Modell, das Vier-Ohren-Modell von Schulz von Thun. Ebenfalls nützlich sind Change Management, wie z.B. (Rising u. Manns, 2004), sowie ein Grundstock an psychologischem Grundwissen, wie beispielsweise in (Vigenschow u. Schneider, 2007) beschrieben.

Nicht alle dieser eigentlich wünschenswerten Kompetenzen können in der zur Verfügung stehenden Lehr- und Lernzeit im vollen Umfang vermittelt werden. In den Kompetenzbereichen, die wir nicht vollständig bedienen können, soll unsere Ausbildung zumindest das Bewusstsein für die Notwendigkeit dieser Kompetenzen wecken, indem sie deutlich macht, für welche Aufgaben, Disziplinen und Rollen diese Kompetenzen erforderlich sind.

### Lehrkonzept

Um dem Curriculum der von uns bedienten Bachelor-Studiengänge „Informatik“ und „Wirtschaftsinformatik“ mit nur einem Beispiel gerecht zu werden konzipierten wir dieses als eine moderne Webapplikation. Bislang kommt dieses Beispiel in folgenden Modulen zum Einsatz, deren Umfang jeweils bei zwei SWS seminaristischem Unterricht und zwei SWS Praktikum liegt und mit fünf ECTS-Punkten gewichtet ist:

- Modul „Software Engineering I“ im 3. Semester des Bachelorstudiengangs Wirtschaftsinformatik
- Modul „Software Engineering II“ im 4. Semester des Bachelorstudiengangs Wirtschaftsinformatik
- Modul „Software-Architektur“ im 4. Semester des Bachelorstudiengangs Informatik

Die zu Größe der zu unterrichtenden Gruppen liegt bei etwa 45 Studierenden. Für das Praktikum werden die Gruppen jeweils noch einmal geteilt.

Thematisch bildet das Beispiel ein Verleihsystem namens *ShareIt* ab, über das ein Freundes- und Bekanntenkreis dingliche Ressourcen unterschiedlicher Art (wie z.B. Bücher, DVDs, Werkzeug, etc.) untereinander verleihen bzw. gemeinschaftlich nutzen kann. Technisch ist die Beispielapplikation als klassische drei-Schicht-Architektur realisiert. Fachlich gliedert sich das Beispiel bisher in die Hauptbereiche „Benutzerverwaltung“, „Verwaltung der Leihobjekte“ und „Ausleihe“ (siehe Abbildung 1).

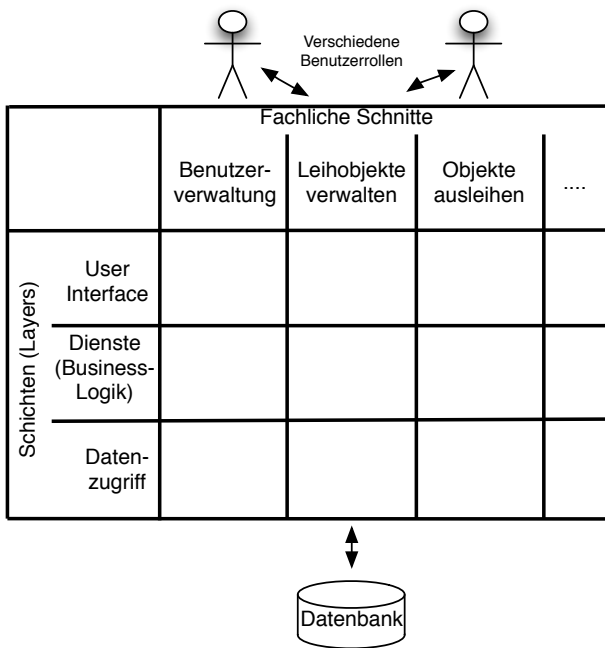


Abbildung 1: Struktur des durchgängigen Beispiels

Für den ersten fachlichen Schnitt, die Benutzerverwaltung, werden Planung, Analyse- und Entwurfsdokumente, Implementierung und umfangreiche Tests den Studierenden als Lernmaterialien zur Verfügung gestellt. Diese dienen sowohl als veranschaulichendes Beispiel als auch als Vorlage für die von den Studierenden durchzuführenden Projektarbeiten.

Der zweite fachliche Schnitt „Verwaltung der Leihobjekte“ ist ebenfalls ausgearbeitet, je nach Fokus der Lehrveranstaltung aber nur teilweise oder gar nicht für die Studierenden verfügbar. In der Lehrveranstaltung „Softwarearchitektur“ (Bachelor Informatik) erhalten die Studierenden die vorgefertigten Analyse- und Entwurfsdokumente, die sie anschließend in eine funktionsfähige Implementierung umsetzen. In „Software Engineering“ (Bachelor Informatik und Wirtschaftsinformatik) dagegen erstellen die Studierenden selbst die benötigten Analyse- und Entwurfsdokumente und setzen anschließend ihre eigene Spezifikation um. Bei Bedarf ist das Beispiel um zusätzliche fachliche Schnitte erweiterbar.

Wir setzen das Lehrmaterial in den einzelnen Veranstaltungen wie folgt ein:

**Software Engineering I:** Hier stehen die Kompetenzfelder Requirements, Design sowie Engineering Management und Process im Vordergrund. Vorgabe für das Praktikum ist daher lediglich eine ausgearbeitete Spezifikation für den ersten fachlichen Schnitt „Benutzerverwaltung“. Die Studierenden müssen mindestens einen weiteren fachlichen Schnitt selbstständig spezifizieren.

**Software Engineering II:** Dieses Modul deckt den Kompetenzbereich Construction ab. Hier müssen die

Studierenden den im ersten Teil spezifizierten fachlichen Schnitt implementieren. Als Vorgabe erhalten sie eine referenzimplementierung des ersten Schnittes.

**Software-Architektur:** Lernziel ist hier die Fähigkeit, moderne Architekturen für komplexe Software-Systeme zu bewerten, zu entwerfen, zu realisieren und zu betreiben (kompetenzbereiche design. Construction, Testing). Die Studierenden bekommen für den ersten fachlichen Schnitt sowohl eine Spezifikation als auch deren Implementierung als Vorgabe. Sie müssen den zweiten fachlichen Schnitt implementieren und bekommen dazu die Spezifikation als Vorgabe.

Als Lehrmethode im seminaristischen Unterricht wurde eine Mischung aus interaktiver Vorlesung, Lernen durch Lehren, Gruppenpuzzle und Projektarbeit in wechselnden Teams eingesetzt.

## Ableich Lehrkonzept vs. Lernziele

Nach den bisherigen Erfahrungen adressiert unser Lehrkonzept eine Vielzahl der Schlüsselkompetenzen, die aus der Praxis gefordert wurden.

## Sachkompetenz

Die Studierenden durchlaufen im Rahmen der Projektarbeit mindestens einen kompletten Softwareentwicklungszyklus, vom Skizzieren erster Anforderungen bis hin zum Integrationstest. Das vorgefertigte, durchgängige Beispiel hilft dabei, neu eingeführte Techniken und deren zielgerichtete Verwendung zu veranschaulichen. Des Weiteren dient es als Anhaltspunkt für die Lösungsbestandteile, welche die Studierenden im Rahmen ihrer Projektarbeit nach und nach selbst entwickeln.

Das Beispiel ist aktuell auf der Basis von Servlets implementiert. Als Frameworks kommen Hibernate, Google Guice (Dependency Injection), Apache Click, Log4j und Selenium zum Einsatz. Die zugehörigen Technologien und Frameworks werden in der Lehrveranstaltung sukzessive vermittelt und von den Studierenden bei der Erstellung ihrer eigenen Lösungsanteile entsprechend weiterverwendet.

Für die Modellierung stehen verschiedene Modellierungswerkzeuge zur Verfügung, beispielsweise der IBM Rational Software Modeler oder die UML2-Modellierungsumgebung der aktuellen Eclipse-Version. Entwickelt wird mit dem JEE-Plug-in-Set von Eclipse.

Die Software des vorgefertigten Beispiels wird den Studierenden in einem Subversion-Repository zur Verfügung gestellt. Über dieses Repository koordinieren die Studierenden den Austausch der im Team erstellten Artefakte. Des Weiteren geben sie darüber ihre Ergebnisse ab.

Im Rahmen der Projektarbeit experimentieren die Studierenden mit verschiedenen Vorgehensmodellen. Neben einem klassischen, iterativ-inkrementell orien-

tierten ingenieurmäßigen Vorgehen kommt als Repräsentant der agilen Ansätze auch Scrum zum Einsatz.

## **Methodenkompetenz**

Um die Studierenden nicht bereits am Anfang mit einer großen Vielzahl verwendeter Technologien zu überfordern, dient ein minimalistisches Exzerpt dieses Beispiels als niederschwelliger Einstieg für den Lernprozess. So wird an einem zunächst noch bewusst sehr einfach gehaltenen System zunächst ein Überblick über die wesentlichen Zusammenhänge zwischen Modellen und Implementierung, sowie innerhalb der Implementierung über das Zusammenspiel der einzelnen Technologien erarbeitet. Darauf aufbauend wird dieses System sukzessive weiter ausgebaut und nach und nach um entsprechende Komplexitäten erweitert, nicht nur bzgl. der realisierten Funktionalität, sondern auch bzgl. der verwendeten Technologien.

Dadurch, dass das Beispiel hinreichend komplex ist und die Aufgabenstellung nicht alle Entscheidungen vorgibt, müssen die Studierenden an verschiedenen Stellen abwägen und entscheiden, wie weit sie bestimmte Kerntechniken anwenden (beispielsweise hinsichtlich des Detaillierungsgrades bei der Modellierung, oder bzgl. der Testabdeckung). Da auch innerhalb des studentischen Projektes Zeit eine knappe Ressource darstellt, wird die Notwendigkeit der Abwägung zwischen Perfektionismus und pragmatischer Einschränkung des investierten Aufwandes erlebbar.

Die Ergebnisse, die die Studierenden im Rahmen der praktischen Projektarbeit erstellen, umfassen neben dem eigentlichen Quelltext auch weitere Artefakte des Software Engineerings, beispielsweise Projektpläne, Anforderungsdokumente, Architekturmodelle und Ähnliches. Da diese Dokumente in die Gesamtbewertung mit einfließen, spendieren die Studierenden zwar meist ein gewisses Maß an Sorgfalt in deren Erstellung, sehen diese Dokumente zunächst aber oft als ein notwendiges Übel an. Das Verständnis für die fachliche Notwendigkeit dieser Dokumente (und damit eine intrinsische Motivation, diese Dokumente sorgfältig zu erstellen und weiter zu pflegen) erwächst im Laufe der Lehrveranstaltung aus der Dauer und dem Umfang der Projektarbeit. Typischerweise stellen die Studierenden nach wenigen Wochen fest, dass ihr Projekt ohne klare Dokumentation ineffektiv wird, weil Anforderungen unklar und doppeldeutig sind, bereits getroffene Entscheidungen nicht mehr einheitlich erinnert werden oder weil Uneinigkeit darüber besteht, wer wann welche Schritte als nächstes durchzuführen hat.

## **Selbstkompetenz**

Im Rahmen der verschiedenen Phasen der Projekt- und Teamarbeit erhalten die Studierenden Feedback von Kommilitonen und Dozenten, sowohl über ihren Arbeitsprozess als auch über die erzielten Ergebnisse. Beispielsweise stellen einzelne Projektteams Teile ihrer Arbeiten den anderen Gruppen vor. Ziel dabei

ist das Erlernen des konstruktiven fachlichen Austausches, sowohl in der Rolle des Präsentierenden als auch in der Rolle des kritisch Hinterfragenden. Beide Rollen fallen erstaunlich vielen Studierenden zunächst schwer. Eine wichtige Voraussetzung für die Vermittlung von Reflexions- und Kritikfähigkeit besteht darin, in der Lehrveranstaltung eine Atmosphäre der vertrauensvollen Zusammenarbeit zu schaffen, in der alle Beteiligten auf Augenhöhe wertschätzend miteinander umgehen.

Bzgl. der Vermittlung ethischen Handelns stoßen wir mit der Ausbildung in dieser Form schnell an Grenzen. Als Dozenten können wir hier im Wesentlichen sensibilisieren, den Blickwinkel auf grundlegende Fragestellungen richten und natürlich versuchen, als gutes Beispiel voranzugehen. Die hinter dem ethischen Handeln liegenden Wertesysteme lassen sich „mal eben so nebenbei“ aber nur ansatzweise vermitteln – und auch nur dann, wenn die der Hochschule vorgelagerte Erziehung und Bildung den Grundstock dafür gelegt hat.

## **Sozialkompetenz**

Durch den relativ hohen Anteil an Teamarbeit kommt dem Bereich der Sozialkompetenz quasi als „Hintergrundprozess“ eine fortlaufend wichtige Rolle zu. Bereits beim Bilden der Teams und bei der Aufteilung der zu erledigenden Arbeiten treffen unterschiedliche Interessen und Vorlieben aufeinander. Die Dozenten beobachten aufmerksam, aber unaufdringlich das Zusammenspiel innerhalb der einzelnen Teams. Bei Bedarf greifen sie (möglichst minimalistisch) in das Geschehen ein, beispielsweise durch das Setzen von gruppenspezifischen Impulsen. Des Weiteren moderieren sie zu ausgewählten Meilensteinen der Projektarbeit den Prozess der Gruppenreflexion über Ergebnisse und Ablauf der Teamarbeit.

Ergänzend wird theoretisch fundiertes Grundlagenwissen zu Sozialkompetenzen in spezialisierten Veranstaltungen vermittelt und gezielt praktisch eingeübt.

## **Zusammenfassung und Ausblick**

Mit dem hier vorgestellten Beispiel lassen sich viele zentrale Aspekte des Software Engineerings von der Machbarkeitsanalyse bis hin zum Abnahmetest zeigen. Der besondere Fokus liegt dabei auf der Integration der einzelnen Disziplinen. Dadurch werden für die Lernenden die Zusammenhänge zwischen einzelnen Arbeitsschritten und Artefakten klar ersichtlich und nachvollziehbar. Insbesondere werden dadurch die Auswirkungen einzelner Entwicklungsentscheidungen greifbar verdeutlicht. Das Beispiel und die zugehörigen Lernmaterialien ermöglichen so einen ganzheitlichen Einstieg in die komplexe Materie des Software Engineerings.

Im Rahmen einer qualitativen Evaluierung haben unsere Studierenden die folgenden repräsentativen Aussagen gemacht. Als positiv wurde die Praxisnähe

des Beispiels empfunden, sowie die Tatsache, dass in den Lehrveranstaltungen ein Beispiel von realistischer Komplexität behandelt wird. Diese Komplexität wurde jedoch gleichzeitig auch als negativ empfunden, da zu Beginn der Veranstaltung sehr viele Dinge gleichzeitig zu lernen sind, beispielsweise verschiedene Frameworks. Des Weiteren wurde die Größe der Aufgabe kritisiert.

Der beschrittene Weg, für ein signifikant großes Projekt einzelne Teile in einer ausgearbeiteten Form vorzugeben, die auch kritischen Blicken aus der Praxis Stand hält, hat sich also im Ansatz bewährt. An manchen Stellen haben wir einen Teil unserer Studierenden mit der Komplexität des Beispiels aber offenbar überfordert.

In folgenden Punkten haben wir konkret Verbesserungspotenzial sowohl für das Beispiel als auch für die eingesetzten didaktischen Methoden identifiziert:

- *Kleinere Wissensseinheiten vermitteln*  
Die Darstellung von Technologien und Basistechniken müssen wir gezielt in einem überschaubaren Kontext aufbereiten.
- *Referenzimplementierung inkrementell einführen*  
Wir werden Referenzimplementierungen zukünftig in kleineren Zwischenschritten vorgeben. Zum Einstieg haben wir ein minimalistisches Beispiel gebaut, das sich auf einen Server (Tomcat, Jetty) „deployen“ lässt und einen Login für einen hart codierten Benutzer mit entsprechendem Service, aber ohne eine Datenbank, realisiert.
- *Mehr Diskussion individueller Lösungen*  
Wir wollen mit den Studierenden stärker deren Lösungen diskutieren. Hier stoßen wir allerdings von der Betreuungsrelation und unserem Zeitbudget an Grenzen.

## Literatur

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

[Evans 2003] EVANS, Eric: *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Longman, Amsterdam, 2003

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

[Martin 2008] MARTIN, Robert C.: *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall International, 2008

[Rising u. Manns 2004] RISING, Linda ; MANN, Mary L.: *Fearless Change: Patterns for Introducing New Ideas*. Addison-Wesley, 2004

[Schaeper u. Briedis 2004] SCHAEPER, H. ; BRIEDIS, K.: *Kompetenzen von Hochschulabsolventinnen und Hochschulabsolventen, berufliche Anforderungen und Folgerungen für die Hochschulreform*. HIS-Kurzinformation, 2004

[Schott u. Ghanbari 2009] SCHOTT, F. ; GHANBARI, S. A.: Modellierung, Vermittlung und Diagnostik der Kompetenz kompetenzorientiert zu unterrichten – wissenschaftliche Herausforderung und ein praktischer Lösungsversuch. In: *Lehrerbildung auf dem Prüfstand 2* (2009), Nr. 1, S. 10–27

[Spitzer 2006] SPITZER, Manfred: *Lernen: Gehirnforschung und die Schule des Lebens*. Spektrum Akademischer Verlag, 2006

[Vigenschow u. Schneider 2007] VIGENSCHOW, Uwe ; SCHNEIDER, Börn: *Soft Skills für Softwareentwickler – Fragetechniken, Konfliktmanagement, Kommunikationstypen und -modelle*. dpunkt.verlag, 2007