Dissertation

Interdisziplinäre Fachdidaktik für Software Engineering – Forschungsbasierte Entwicklung und Evaluation eines anwendungsbezogenen didaktischen Ansatzes

Inaugural-Dissertation

in der Fakultät Humanwissenschaften

der Otto-Friedrich-Universität Bamberg

vorgelegt von

Yvonne Sedelmaier

aus

Lichtenfels

Bamberg, den 10.12.2015

Tag der mündlichen Prüfung: 2. Februar 2016

Dekan/Dekanin: Universitätsprofessor Dr. Stefan Hörmann

Erstgutachter/-in: Universitätsprofessor Dr. Walter Bender

Zweitgutachter/-in: Professor Dr. Ralf Reißing

Kurzfassung

Software zieht sich mittlerweile durch alle Bereiche des täglichen Lebens. Mit der Bedeutung von Software wächst auch die Bedeutung der Ausbildung derer, die diese Software trotz aller Komplexität qualitativ hochwertig entwickeln können. Jedoch fehlten bisher methodisch fundierte Aussagen, was alles zu einer guten Software-Engineering-Ausbildung gehört.

Diese Arbeit entwirft ein Forschungsdesign, das zu einer interdisziplinären Fachdidaktik für Software Engineering führt. Ausgehend von Kompetenzprofilen *guter* Software-Ingenieure werden dabei Maßnahmen für die Hochschullehre abgeleitet, um diese Kompetenzziele zu erreichen. Diese Maßnahmen sind schließlich kompetenzorientiert zu evaluieren.

Kernelemente des Forschungsansatzes umfassen also ein Kompetenzprofil, die darauf aufbauende kompetenzorientierte Weiterentwicklung von Lehrkonzepten sowie die systematische Evaluation der Wirksamkeit dieser Lehrkonzepte, um daraus Rückschlüsse auf die Lehr-Lern-Prozesse und ihre Wirkzusammenhänge zu ziehen.

Das Ergebnis der gewonnenen Erkenntnisse ist ein Vorschlag für eine interdisziplinäre Fachdidaktik für Software Engineering.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation und Zielsetzung

1.1 Software Engineering

Software beeinflusst mittlerweile viele Bereiche unseres Alltags. Viele Geräte werden mittels Software gesteuert, angefangen vom Herzschrittmacher, über Waschmaschinen und Airbags in Autos bis hin zu Mobiltelefonen (vgl. Sommerville, 2011). Darüber hinaus übernehmen neue Medien und mobile Geräte wie das Internet und Smartphones immer mehr Funktionen in unserem Leben. Diese Geräte werden erst durch Software nutzbar. Wir buchen Fahrkarten über das Handy, rufen E-Mails mobil ab und spielen gegen andere Online-Spieler, die über die gesamte Welt verteilt sind, was ohne Software nicht möglich wäre. Diese zunehmende Technisierung unserer Welt führt dazu, dass Software immer mehr an Bedeutung gewinnt. In diesem Zuge kommt auch der Ausbildung im Software Engineering als Querschnittsdisziplin vermehrt Bedeutung zu.

Software Engineering beschäftigt sich mit der Entwicklung komplexer Softwaresysteme in einem großen, interdisziplinären Team über einen längeren Zeitraum hinweg und für eine mehr oder weniger bekannte Gruppe von späteren Anwendern. Häufig kann Software nicht nur von einem einzelnen Programmierer[1] entwickelt werden. Softwareentwicklung findet meist in einem oder mehreren Teams von Entwicklern statt. Dadurch werden zwar die Aufgabenpakete der einzelnen Entwicklungsteams überschaubarer und somit auch beherrschbarer gemacht. Jedoch steigt der Koordinationsaufwand zwischen den beteiligten Entwicklerteams nicht-linear. Software-Ingenieure kennen dieses Phänomen im technischen Kontext als den Zusammenhang zwischen Kohäsion und Kopplung (vgl. Yourdon & Constantine, 1979).

Gerade auch in der Komplexität von Softwareentwicklungsprojekten liegen viele Tücken und Herausforderungen. Man denke nur an das vom amerikanischen Präsidenten Obama initiierte Gesundheitssystem „ObamaCare“, dessen Einführung mit großen Schwierigkeiten behaftet war, was daran lag, dass zahlreiche Softwarefirmen an der Entwicklung der neuen Registrierungsplattform beteiligt und kaum zu koordinieren waren (vgl. Schmitt, 2013). Als ein weiteres Beispiel für ein umfangreiches Software-Entwicklungsprojekt kann Toll Collect angeführt werden. Das Mautsystem konnte aufgrund technischer Probleme erst mit 16 Monaten Verspätung an den Start gehen (vgl. Schwenn, 2005). Der Bund machte in einem Schiedsverfahren für diese Zeit 3,5 Milliarden Euro Einnahmeausfälle geltend. Als ein wesentlicher Grund für die Verzögerung werden Softwareprobleme genannt (vgl. Brychcy, Büschemann &

[1] Im Sinne einer besseren Lesbarkeit beschränkt sich das Dokument bei der Nennung von Personen, Berufen, Positionen und Titeln auf die männliche Form, ohne damit die weibliche ausschließen zu wollen.

Rubner, 2003; Haustein-Teßmer, 2003). Auch wenn der Zusammenhang zwischen Projektgröße und -erfolg nicht zweifelsfrei belegbar ist (vgl. The Standish Group International, 1999), scheitern viele Entwicklungsprojekte an zu engen Budget- und Zeitvorgaben oder an Änderungen von Zielsetzung und Anforderungen (vgl. Emam & Koru, 2008).

Softwareentwicklung ist ein sehr komplexer Prozess, der weit über das reine Programmieren hinausgeht. So arbeiten Informatiker in der Software Entwicklung etwa als Anforderungsanalytiker, Softwarearchitekten oder Tester. Softwareentwicklung umfasst also zahlreiche Aktivitäten, die zum einen dazu führen, dass es innerhalb eines Entwicklungsteams eine Vielzahl an Rollen gibt, die ausgefüllt werden und zusammenarbeiten müssen (vgl. Höhn, Höppner & Rausch, 2008, S. 113-114; Pfleeger & Atlee, 2006, S. 95). Zum anderen erfordert die Entwicklung von Software auch Kommunikation mit zahlreichen Stakeholdern und Kunden außerhalb des Teams (vgl. Pfleeger & Atlee, 2006, S. 97; Vigenschow, Schneider & Meyrose, 2011). „Drei Viertel ihrer Arbeitszeit benötigen Software-Entwickler für die Kommunikation mit verschiedenen Partnern: Auftraggebern, Benutzern, Kollegen, Management, Vertrieb, usw.“ (Funken, 2001, S. 48)

Hinzu kommt, dass Software im Normalfall eben gerade nicht für Informatiker entwickelt wird, sondern für Anwender aus nahezu allen anderen Fachbereichen. Um in diesem Umfeld zu bestehen, benötigen Software-Ingenieure neben fundiertem Fachwissen auch zahlreiche überfachliche Kompetenzen (vgl. Funken, 2001, S. 12-13). Erst diese sogenannten „weichen Kompetenzen“ ermöglichen es Software-Ingenieuren, ihr fachliches Können in die Tat umzusetzen. Reines Fachwissen alleine genügt dafür nicht.

Zusammenfassend lässt sich festhalten, dass Software Engineering eine Reihe von Besonderheiten aufweist, denen bereits in der Ausbildung Rechnung getragen werden muss: Neben umfassendem fachlichen Können sind besonders interpersonelle, überfachliche Kompetenzen von Bedeutung. Entgegen der weit verbreiteten Vorstellung, dass ein Informatiker alleine im dunklen Keller sitzt und programmiert, sind gerade im Software Engineering ausgeprägte kommunikative Fähigkeiten gefragt (vgl. Funken, 2001, S. 46-47). Auch die Zusammenarbeit in einem Team nimmt einen hohen Stellenwert ein und Software-Ingenieure müssen sich in sehr große Teams von z.B. hundert Personen einfügen können. Ebenso ist es notwendig, in der inhärenten Komplexität den Überblick zu bewahren und so durch Abstraktionsvermögen Software, die ja nicht physisch greifbar ist, mental zu erfassen.

Damit Software-Ingenieure im späteren Berufsalltag bestehen können, ist es also erforderlich, bereits in der Hochschulausbildung sowohl fachliches Können als auch überfachliche Kompetenzen zu adressieren.

1.2 Kompetenzen im Software Engineering

Im Normalfall ist jedoch genau die Komplexität von Software Engineering in der Hochschulausbildung nur bedingt und in Ausschnitten abbildbar. Es sind meist weder hundert Studierende vorhanden, die gemeinsam an einem Softwareprojekt arbeiten könnten, noch Kunden, die selbst nicht so genau ausdrücken können, welche Anforderungen sie an eine zu entwickelnde Software haben, und diese Vorstellung während des Entwicklungsprozesses sogar noch ändern (vgl. Funken, 2001, S. 23). Ein Entwicklungsprojekt dauert normalerweise mehrere Monate bis Jahre. Eine Studie über 5.400 IT-Entwicklungsprojekte zeigt die Komplexität von solchen Projekten: Das durchschnittliche Budget der einzelnen Vorhaben lag bei 1,8 Millionen Euro bei einer mittleren Dauer von zwei Jahren, wobei jedes sechste Projekt das vorgegebene Budget um durchschnittlich 200 Prozent sprengte – und das sogar inflationsbereinigt. Der geplante Zeitrahmen wurde in diesen Fällen im Mittel um 70 Prozent überschritten (vgl. Bloch, Blumberg & Laartz, 2012). Im Schnitt überschreiten IT-Projekte den Kostenrahmen um 27 Prozent. Fünf Prozent aller Projekte laufen sowohl finanziell als auch zeitlich aus dem Ruder. Zu ähnlichen Ergebnissen kommen auch die regelmäßig erscheinenden CHAOS-Berichte der Standish Group (vgl. z. B. Eveleens & Verhoef, 2010; The Standish Group International, 1999).

Diese langen Zeitfenster mit den daraus resultierenden Herausforderungen, zum Beispiel sich verändernde Anforderungen und steigender Umfang des Softwareprodukts mit zunehmender Zeitdauer des Projektes, lässt sich im Hochschulkontext nicht abbilden. Es erscheint erforderlich, zunächst einen tragfähigen Grundstock an fachlichem Wissen auszubilden, bevor ein solches Entwicklungsprojekt überhaupt sinnvoll begonnen werden kann, auch wenn einzelne Versuche unternommen wurden, Projekte frühzeitig in die Informatikausbildung zu integrieren (vgl. z. B. Böttcher, Schlierkamp, Thurner & Zehetmeier, 2015). Hinzu kommt, dass es sich bei Softwareentwicklung um einen extrem arbeitsteiligen Prozess mit einer Vielzahl an möglichen fachlichen Rollen und Aufgaben (Anforderungsanalytiker, Softwarearchitekt, Tester, etc.) handelt (vgl. Höhn, Höppner & Rausch, 2008, S. 113-114; Pfleeger & Atlee, 2006, S. 95). Für jeden beteiligten Informatiker ist es eine Herausforderung, den Überblick über das große Ganze zu wahren und durch Abstraktion zu strukturieren (vgl. Funken, 2001, S. 83). Jeder einzelne leistet einen wichtigen Beitrag, muss diesen aber wiederum an vielen Stellen mit

Arbeitsergebnissen anderer vernetzen, diese aufgreifen und weiterentwickeln. Die Arbeit des einzelnen Entwicklers bringt nur dann Mehrwert, wenn sie mit der Arbeit der anderen verbunden wird. So macht es beispielsweise keinen Sinn, sich über die Struktur der zu entwickelnden Software oder sinnvolle Tests Gedanken zu machen, wenn nicht klar ist, was die Anforderungen an das zu entwickelnde System sind. Für Software-Ingenieure ist es manchmal schwierig, den Überblick über das Gesamtsystem zu behalten und den eigenen Beitrag als Teil dessen wahrzunehmen (vgl. Funken, 2001, S. 73).

Da die Mehrzahl der Studierenden auch aufgrund der Bologna-Reformen und der Änderungen im Schulsystem (G8) noch relativ jung ist, fehlt meist persönliche Reife und berufliche Erfahrung. Diese wären aber nötig, um überhaupt ein Problembewusstsein für den Großteil der Herausforderungen im Software Engineering zu entwickeln. So können sich viele Studierende nicht vorstellen, dass die Anforderungen an eine zu entwickelnde Software nicht fix und fertig in einem Lastenheft vorliegen, sondern erst mühsam in einem partizipativen Prozess von fachfremden Kunden erfragt und in eine Form übersetzt werden müssen, mit der im Rahmen der Informatik weitergearbeitet werden kann (vgl. Funken, 2001, S. 12-13). Viele Anwender selbst sehen hier auch kein Problem, denn sie erwarten, dass für die Aufgabenstellung, die sie den Informatikern meist knapp und nicht vollständig schildern, von diesen eine passgenaue Lösung geliefert würde. Vielen Studierenden ist nicht klar, dass beispielsweise Anforderungen sich über eine mehrmonatige Projektdauer mehrfach ändern oder immer neue, auch widersprüchliche, hinzukommen können. Software Engineering erfordert ein hohes Maß an Austausch mit anderen wie z.B. mit späteren Anwendern, die zwar ein „Problem“ haben, es aber weder genau beschreiben können noch eine Lösung dafür kennen. "Darüber hinaus haben die Kunden – in Unkenntnis der (Un-)Möglichkeiten von Softwarelösungen – meist sehr diffuse Vorstellungen von ihren Softwarewünschen und ändern diese häufig im Laufe der wachsenden Vertrautheit mit dem Thema (z.B. durch Prototypendemonstrationen). In der Praxis lässt sich deshalb eine der größten Schwierigkeiten der Entwickler mit der Frage formulieren: Was ist eigentlich das 'Problem'?" (Funken, 2001, S. 23)

Hinzu kommt, dass – selbst wenn Informatiker in der Lage sind, Abläufe im Unternehmen eines Kunden mittels geeigneter Modelle wie z.B. Prozessmodelle abzubilden – sie diese Abläufe zuerst vom Kunden erheben müssen. Häufig werden die Fähigkeiten, um solche Informationen überhaupt zu bekommen, nicht im Studium adressiert. Dazu gehören zum Beispiel Arbeits-, Kreativitäts-, Kommunikations- und Fragetechniken.

Die Kunden haben in der Regel einen anderen, informatikfernen disziplinären Hintergrund. Somit stellen sie Software-Ingenieure vor die Herausforderung, über Disziplingrenzen hinweg zu kommunizieren und sich in die Welt des fachfremden Kunden einzudenken.

Da jede Software unter den exakt gleichen Rahmenbedingungen nur ein einziges Mal entwickelt wird, ist folglich auch jedes Entwicklungsprojekt anders. Für Softwareentwickler bedeutet dies, dass es keine Standardvorgehen oder Standardrezepte gibt. Jedes Entwicklungsprojekt ist einzigartig und erfordert neue, kreative Lösungsansätze.

Auf den Umgang mit genau den eben genannten Herausforderungen sollte eine gute Hochschulausbildung vorbereiten.

2 Notwendigkeit für eine Hochschul-Fachdidaktik für Software Engineering

2.1 Ziele und Herausforderungen in der Hochschullehre von Software Engineering

Bis weit in die 1990er Jahre lag der Schwerpunkt in der Informatik-Hochschulausbildung vorwiegend auf fachlichen Themen. Mittlerweile hat sich jedoch, nicht zuletzt auch aufgrund der Softwarekrise in den 1960er und 70er Jahren, häufig die Erkenntnis durchgesetzt, dass Fachwissen alleine nicht ausreichend für den Berufsalltag im Software Engineering qualifiziert. Eine zunehmende Zahl großer gescheiterter Softwareprojekte löste die Softwarekrise aus. Dijkstra beschreibt bereits 1972 die Ursachen der Softwarekrise folgendermaßen: "The major cause is ... that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem." (Dijkstra, 1972, S. 861). Da seither die Leistungsfähigkeit der Hardware, die Komplexität der Software sowie die Verbreitung entsprechender Geräte mit unbekannten bzw. unzähligen Anwendern noch zugenommen hat, kann die Softwarekrise auch heute noch nicht als überwunden angesehen werden. Um der Softwarekrise und ihren Ursachen entgegenzuwirken, sind im Software Engineering zahlreiche überfachliche Kompetenzen erforderlich, die in einen ingenieurmäßigen methodischen Rahmen eingebettet sind, und die bislang in der technisch-funktional ausgerichteten (Hochschul-)Ausbildung zu wenig adressiert wurden (vgl. Funken, 2001, S. 47). Diese überfachlichen Kompetenzen gilt es jedoch zusammen mit fachlichen Kompetenzen bereits in der Ausbildung von Software-Ingenieuren zu trainieren, auch wenn die Einsicht in deren Notwendigkeit sich erst nach und nach im Berufsbild des Software-Ingenieurs niederschlägt (vgl. Funken, 2001, S. 73).

Eine Herausforderung für die Lehre von Software Engineering liegt im fehlenden Problembewusstsein der Studierenden. Häufig erkennen die Studierenden nicht, welchen Nutzen die im Software Engineering gelehrten Methoden und die dazu erforderlichen Kompetenzen haben sollen. In den ersten Semestern durchlaufen die Studierenden zunächst eine grundlegende Programmierausbildung. In dieser bekommen sie vom Lehrenden immer präzise formulierte Aufgaben, die jeweils ein kleinteiliges, in sich abgeschlossenes Problem adressieren (vgl. Deitel, 2012). Um Programmieren zu lernen, sind diese Aufgaben auch häufig von jedem Studierenden selbst und in Einzelarbeit zu lösen. Für einfache Problemstellungen gibt es eine richtige Musterlösung. Wenn dann ein Lehrender im Software Engineering auf

Studierende mit diesen Vorerfahrungen trifft, ist es für die Studierenden nur sehr schwer nachvollziehbar, warum sie Methoden, Techniken und Kompetenzen lernen sollen, mit denen sie Probleme lösen können, die sie nicht einmal im Ansatz erkennen oder nachvollziehen können. Für die zu lösenden Problemstellungen gibt es aufgrund der Komplexität des Softwareentwicklungsprojekts eine Vielzahl an möglichen Lösungsansätzen, die unter Berücksichtigung der spezifischen Situation ausgewählt, begründet und angewandt werden. Auch wenn kleinteilige Programmieraufgaben die Studierenden zunächst in einer falschen Sicherheit wiegen, vermitteln sie doch für die Umsetzung komplexer Software-Entwicklungs-Projekte erforderliche grundlegende Kenntnisse und Fähigkeiten, um überhaupt erst Software-Entwicklung im Großen angehen zu können. Diese Besonderheiten und Herausforderungen des Software Engineering erschließen sich jedoch meist erst in der beruflichen Praxis. Häufig erleben Informatiker erst im Berufsalltag ihre persönliche Softwarekrise.

Software ist physisch nicht greifbar. Daher setzt Softwareentwicklung auch immer hohes Abstraktionsvermögen voraus, was die Grundlage dafür ist, komplexe Sachverhalte in Modellen abzubilden, zu strukturieren und passende Lösungsideen und -ansätze zu entwickeln (vgl. Funken, 2001, S. 83). Die Fähigkeit zum Umgang mit abstrakten theoretischen Modellen als Denk- und Arbeitsgrundlage muss die Ausbildung im Software Engineering ebenfalls gewährleisten.

Damit geht einher, dass hohe Komplexität nur im Team beherrscht werden kann. In der Softwareentwicklung sind Teams häufig interdisziplinär (vgl. Rupp, 2014, S. 11-12). Innerhalb dieser Teams sind zahlreiche überfachliche Kompetenzen notwendig, um erfolgreich zu sein. Dazu gehört zum Beispiel die Fähigkeit zu kommunizieren und innerhalb des Teams eine gemeinsame Arbeitsweise und -struktur zu entwickeln (vgl. Pfleeger & Atlee, 2006, S. 97). Dies erfordert es, sich auf andere Menschen einzulassen, über den eigenen Tellerrand hinauszublicken und respektvoll miteinander in Interaktion zu treten. Eine weitere Herausforderung liegt darin, mit fachfremden Stakeholdern außerhalb des Teams zusammenzuarbeiten. Software Engineering ist somit hochgradig interdisziplinär. Studierende für die erforderlichen überfachlichen Kompetenzen zu sensibilisieren und diese auch aktiv zu trainieren ist eine zentrale Herausforderung in der Lehre von Software Engineering, um Studierende angemessen auf den Beruf vorzubereiten (vgl. Kabicher, Motschnig-Pitrik & Figl, 2009, S. T1A-2).

Ziel der Hochschulausbildung im Software Engineering heute ist es also, den Studierenden Problembewusstsein und Kompetenzen zum Umgang mit hochkomplexen nicht greifbaren Systemen, die nur über Modelle veranschaulicht und strukturiert werden können, mitzugeben ebenso wie das Wissen, was es bedeutet, Teil eines großen Entwicklungsteams zu sein, und dass es nicht genügt, gut

programmieren zu können. Der Zusammenarbeit mit anderen kommt nun – auch gerade in der Hochschulausbildung von Software-Ingenieuren – eine besondere Bedeutung zu.

Zusammengefasst muss die Lehre von Software Engineering also folgende zentrale Aspekte berücksichtigen:

1. Zuerst gilt es, Problembewusstsein für die Notwendigkeit und Komplexität des Software Engineering sowie ein Verständnis für die Bedeutung von Interdisziplinarität im Software Engineering (vgl. Funken, 2001, S. 73) zu schaffen, um dann
2. die im Kontext von Software Engineering erforderlichen überfachlichen Kompetenzen für den Umgang mit Komplexität und Interdisziplinarität gemeinsam mit den fachlichen zu adressieren.

Mit diesen Aspekten befasst sich die vorliegende Arbeit im Detail. Es wird eine anwendungsbezogene Fachdidaktik für Software Engineering an Hochschulen vorgeschlagen und deren Entwicklung wissenschaftlich fundiert begründet.

Das Vorgehen dieser Arbeit orientiert sich an den von Plöger in Anlehnung an den Strukturplan für das Bildungswesen (*Strukturplan für das Bildungswesen,* 1970) definierten Aufgaben einer Fachdidaktik, in der es darum geht

"1. festzustellen, welche Erkenntnisse, Denkweisen und Methoden der Fachwissenschaft Lernziele des Unterrichts werden sollen;

2. Modelle zum Inhalt, zur Methodik und zur Organisation des Unterrichts zu ermitteln, mit deren Hilfe möglichst viele Lernziele erreicht werden;

3. den Inhalt der Lehrpläne immer wieder daraufhin kritisch zu überprüfen, ob er den neuesten Erkenntnissen fachwissenschaftlicher Forschung entspricht, und gegebenenfalls überholte Inhalte, Methoden und Techniken des Unterrichts zu eliminieren oder durch neue zu ersetzen;

4. erkenntnistheoretische Vertiefung anzuregen und fächerübergreifende Gehalte des Faches beziehungsweise interdisziplinäre Gesichtspunkte zu kennzeichnen." (Plöger, 1999, S. 17)

Auch Kron kommt zu ähnlichen Aufgabenstellungen einer Fachdidaktik, hebt jedoch die „Erarbeitung von Evaluationsverfahren“ (2008, S. 29) hervor, die in dieser Arbeit eine zentrale Rolle spielen.

2.2 Stand der Forschung

Nicht nur in den MINT-Fächern liegt erst in den letzten Jahren, nicht zuletzt aufgrund des einsetzenden Bologna-Prozesses, ein Fokus auf Hochschuldidaktik. Bei diesem Forschungsbereich handelt es sich um ein vergleichsweise neues Gebiet.

Obwohl es Bemühungen in den MINT-Fächern gibt, die Studierendenzahlen zu erhöhen und Abbrecherquoten zu senken, ist im Bereich der Informatik und besonders im Bereich des Software Engineering bisher wenig hochschuldidaktische Forschung in umfassender Form betrieben worden. Es gibt zwar Konferenzen und Publikationen, die sich mit diesem Thema beschäftigen, wie etwa die im zwei-Jahres-Turnus stattfindende Tagung „Software Engineering im Unterricht der Hochschulen (SEUH)“ (vgl. z. B. Ludewig & Böttcher, 2011). Jedoch dominiert hier häufig die Sicht der Ingenieurwissenschaften ohne belastbares pädagogisch-didaktisches Fundament. Dieses Phänomen der „Pädagogisierung“ ohne Berücksichtigung pädagogischen Fachwissens beobachtet auch Achtenhagen (2006, S. 590). Bisher liegt also keine Fachdidaktik für Software Engineering vor. Da Software Engineering als Teilbereich der Informatik gesehen werden kann, scheint es naheliegend, zunächst im Bereich der MINT-Didaktiken nach übertragbaren Ansätzen zu suchen.

Bisherige fachdidaktische Fragestellungen, die sich auf Informatik beziehen, beschäftigen sich entweder mit Fachdidaktik der Informatik wie beispielsweise die in zweijährigem Turnus stattfindende Fachtagung „Hochschuldidaktik der Informatik“ (vgl. z. B. Schmolitzky, Rick & Forbrig, 2013), welche ein viel größeres und anderes Lehrgebiet – nämlich die Informatik – zu Grunde legt, oder aber mit didaktischen Fragestellungen für den Schulunterricht (vgl. Hubwieser, 2007), was eine völlig andere Zielgruppe adressiert. Zudem hat Schulunterricht nicht die Absicht, die Komplexität von Software Engineering abzubilden. Vielmehr werden hier Modellierungs- und Strukturierungsaspekte alleine betrachtet und adressiert. "Das übergeordnete Ziel informatischer Bildung in Schulen ist es, Schülerinnen und Schüler bestmöglich auf ein Leben in einer Informationsgesellschaft vorzubereiten, das maßgeblich durch den verbreiteten Einsatz von Informations- und Kommunikationstechnologien […] geprägt ist. Jede Schülerin und jeder Schüler soll dazu in die Lage versetzt werden, […] den grundlegenden Aufbau von »Informatiksystemen« und deren Funktionsweise zu verstehen, um damit einerseits deren zielgerichtete Anwendung bei der Lösung von Problemen, aber auch die leichte Erschließung anderer Systeme der gleichen Anwendung zu ermöglichen." (Puhlmann, 2008, S. 11) In diesem Zitat wird deutlich, dass im Schulunterricht zwar Abstraktionsvermögen nicht vernachlässigt werden kann, jedoch häufig die reine Anwenderperspektive eingenommen wird und Informatik in der Schule den Umgang mit bestehenden Informations- und Kommunikationstechnologien

adressiert. Software Engineering befasst sich im Gegensatz dazu jedoch mit der Entwicklung komplexer Softwaresysteme.

Didaktische Forschungen in den Naturwissenschaften dagegen zielen häufig auf die Beseitigung inhaltlicher bzw. sachlicher Fehlvorstellungen der Lernenden und sind meist auf schulische Ausbildung konzentriert (vgl. z.B. Kattermann, Duit, Gropengießer & Komorek, 1997; Kircher, 2015). Falsche Annahmen der Lernenden, beispielsweise dass Strom „fließt“, verhindern, dass diese naturwissenschaftlichen Phänomene erklärt werden können (vgl. Hank, 2014). Haben die Studierenden falsches „Vorwissen“, sind diese Erklärungen nicht mehr stimmig und können so kaum mehr verstanden und somit nur schwer gelernt werden. Hinzu kommt, dass Naturwissenschaften einem nomothetischen Grundansatz folgen, der darauf abzielt, die Welt durch Gesetzmäßigkeiten zu erklären (vgl. Erpenbeck, 2012, S. 8). Im Software Engineering gibt es jedoch keine Gesetzmäßigkeiten und Regeln, die verlässlich immer funktionieren und dann wie eine Formel angewandt werden könnten, da jedes Projekt anders ist und vorhandene Lösungskonzepte auf jede spezielle Entwicklungssituation angepasst werden müssen. Daher können die fachdidaktischen Konzepte der Naturwissenschaften nicht oder nur sehr eingeschränkt auf Software Engineering übertragen werden.

Die Didaktik der Mathematik fokussiert ähnlich wie die Didaktik der Informatik auf den klassischen Schulunterricht (vgl. z.B. Jungwirth, 2014, S. 9; Leuders, 2011). Fachdidaktische Fragestellungen in der Mathematik beschäftigen sich zudem häufig mit der Legitimation des Fachs Mathematik an sich (vgl. z. B. Maaß, 2007). Eine kontroverse Diskussion in der Mathematikdidaktik dreht sich darum, ob das vermeintlich wesentliche und in der Praxis benötigte mathematische Wissen in den ersten sieben Schuljahren gelernt wird und diskutiert insbesondere den direkten Anwendungsbezug der höheren Mathematik (Heymann, 1996). Doch auch der elementaren Mathematik gelingt es in der Praxis häufig nicht, diesen Praxisbezug herzustellen und „die Verbindung zwischen der Mathematik und dem 'Rest der Welt'“ herzustellen (Reiss & Hammer, 2012, S. 9). „So wundert es auch nicht, wenn die Schülerinnen und Schüler eine unüberbrückbare Kluft zwischen ihrem Leben in der Schule und dem Leben außerhalb sehen. Ein Vorschlag der Mathematikdidaktik zur Überwindung des Konflikts liegt darin, in eher künstliche Kontexte eingekleidete Sachaufgaben deutlich als solche erkennbar zu machen und wenigstens hin und wieder *echte* Probleme aus dem Alltag lösen zu lassen. Geeignete Probleme müssen identifiziert und im Unterricht schülergerecht thematisiert und aufbereitet werden.“ (Reiss & Hammer, 2012, S. 9). Herausforderung in der Mathematikdidaktik (zumindest an Schulen) scheint also die In-Bezug-Setzung mathematischer Probleme mit der Lebenswirklichkeit der Schüler zu sein. Dieser Herausforderung versucht die Didaktik

der Mathematik mit entsprechenden Aufgabenstellungen, die näher am Alltag der Lernenden sind, zu begegnen (vgl. Leuders, 2009). Diese Ansätze der Mathematikdidaktik zielen auf die einfache, elementare Mathematik und sind sehr stark auf den frühkindlichen bzw. schulischen Bereich bezogen (vgl. Steinweg, 2013). Für höhere Mathematik, wie sie im Hochschulkontext Anwendung findet, existieren aktuell keine tragfähigen fachdidaktischen Forschungen (vgl. Jungwirth, 2014, S. 9). Dies wäre jedoch der Bereich, in dem auf Software Engineering übertragbare Ansätze zu finden sein könnten, da sowohl für höhere Mathematik als auch für Software Engineering das Problembewusstsein bei den Studierenden und damit die Antwort auf die Frage, wozu sie das lernen und verstehen sollten, häufig fehlt (vgl. Heymann, 1996). Beide Fachgebiete sind nur schwer zugänglich und abstrakt. Einzig die Berechenbarkeit der Mathematik ist nicht auf Software Engineering übertragbar, da – wie bereits bei den Naturwissenschaften erläutert – es keine allgemeingültigen Regeln und Gesetze gibt, die eins zu eins zur Lösung eines Problems angewandt werden können. Anders als die höhere Mathematik weist Software Engineering einen hohen Praxis- und Anwendungsbezug auf, auch wenn die Studierenden eben diesen noch nicht erkennen können.

Die Technikdidaktik ist entweder sehr vom Berufsschulunterricht geprägt und stark auf gewerblich-technische Ausbildungsberufe (vgl. Ott, 2000) oder die technisch orientierte Ausbildung etwa von Ingenieuren wie Maschinenbauern ausgerichtet. Technik-didaktische Konzepte fokussieren sehr stark auf den Lernraum (vgl. z.B. Bernard, 2001; Fast, 2006) und damit verbunden die Methodenwahl, die eine starke Handlungsorientierung aufweist (vgl. Bleher, 2001, S. 294-295). In Lernräumen, also Laboren, werden Versuche aufgebaut, in denen berechenbare, vorhersagbare Zusammenhänge veranschaulicht werden. Dabei greift die Technikdidaktik häufig auf Experimente sowie weitere problemorientierte Lernansätze zurück (vgl. Hüttner, 2005). Da die Lehrziele häufig auf die praktische Handhabung von Werkzeugen, Maschinen, Geräten etc. sowie auf das Sammeln von Materialerfahrungen ausgerichtet sind (vgl. Bleher, 2001, S. 295), werden als wesentliche Methoden etwa Konstruktions- und/oder Herstellungsaufgaben, technische Experimente, technische Analysen, Erkundungen im Rahmen des Technikunterrichts und technische Bewertungen (vgl. Henseler, 1996) genannt. Die starke Fokussierung auf inhaltsbezogene Lernziele im Technikunterricht hat also ein eingeschränktes Methodenrepertoire zur Folge, „welche schwerpunktmäßig auf die Vermittlung von Kenntnissen, Fähigkeiten und Fertigkeiten bei der Herstellung von Produkten abzielen." (Bleher, 2001, S. 296) Es lässt sich also zusammenfassen, dass Technikdidaktik sehr stark auf berechenbaren physikalischen oder mathematischen Ansätzen fußt und sich somit deutlich vom Software Engineering unterscheidet.

So hat jeder MINT-Bereich mit seinen eigenen Herausforderungen zu kämpfen und die Konzepte können nur schwer auf jeweils andere Fächer übertragen werden.

2.3 Charakteristika des Fachs Software Engineering

Software Engineering nimmt innerhalb des MINT-Bereichs eine besondere Stellung ein. Es ist keine klassische Naturwissenschaft oder rein technische Disziplin mit berechenbaren Grundlagen und zudem auch noch eine Querschnittsdisziplin, die in vielen anderen Bereichen in verschiedenen Ausprägungen Anwendung findet (vgl. Gesellschaft für Informatik e.V. (GI), 2006).

Zusammenfassend lässt sich also sagen, dass im Software Engineering, anders als in den klassischen MINT-Fächern, eine starke Kompetenzorientierung vorliegt und weniger die Anhäufung reinen „handwerklichen“ Wissens im Vordergrund steht. Dadurch gewinnen überfachliche Kompetenzen im Software Engineering im Unterschied zu sehr „wissenslastigen“ Disziplinen stark an Bedeutung, was sich auch in der Ausbildung niederschlägt. Im Gegensatz zu Mathematik, Naturwissenschaften oder den darauf aufbauenden technischen Bereichen kann Software Engineering aufgrund fehlender Gesetzmäßigkeiten und allgemeingültiger Regeln nicht in Versuchen beliebig oft in unveränderter Form reproduziert werden. Es gibt eben keine solchen Gesetzmäßigkeiten, die immer gelten und sicher zum Erfolg führen. Hinzu kommt, dass Software nicht physisch greifbar ist.

Anders als in den anderen MINT-Bereichen ist der Ablauf im Software Engineering nicht berechenbar und somit nicht wiederholbar. Kein Projekt findet zwei Mal in derselben Form statt. Selbst wenn dieselbe Software zweimal mit denselben Beteiligten im selben Umfeld entwickelt würde, wäre der Projektverlauf und das Ergebnis deutlich anders als beim ersten Mal, da eine starke menschliche Komponente beteiligt ist und Menschen täglich durch Erfahrungen dazulernen, das Umfeld nicht identisch ist und eine Vielzahl an möglichen Lösung dazu führt, dass der eingeschlagene Lösungsweg und somit das Ergebnis nicht identisch zum ersten Mal sein können. Der Erfolg im Software Engineering hängt maßgeblich davon ab, ob die beteiligten Entwickler in der Lage sind, sich immer wieder in die komplexe Welt des Kunden oder Anwenders einzudenken. "Aushandlungsprozesse zwischen Partnern, die aus unterschiedlichen Handlungskontexten stammen, verlangen von den Beteiligten jedoch die Fähigkeit zum Perspektivenwechsel und die Anerkennung gleichwertiger Entscheidungskompetenzen." (Funken, 2001, S. 15). Auch Lerch (2014, S. 87) stellt fest, dass die eigenen Denkstile eine Herausforderung in interdisziplinären

Projekten und Arbeitsformen sind und dass diese Denkstile maßgeblich durch die jeweilige fachliche Disziplin geprägt werden.

Problemstellungen im Software Engineering sind anders als in den restlichen MINT-Fächern eher mit den Herausforderungen in den Humanwissenschaften und besonders der Pädagogik vergleichbar, in der, ohne klare Ursache-Wirkungsbeziehungen definieren zu können, im Lösungsprozess immer wieder auf komplexe unvorhersagbare Aktionen komplexer „Systeme“ planvoll reagiert werden muss. Auch Funken hebt die Sonderstellung des Software Engineering in den MINT-Fächern hervor (vgl. Funken, 2001, S. 136).

Auch wenn in MINT-Fächern das vorherrschende nomothetische Grundverständnis langsam aufweicht, so prägt es das Grundverständnis dieser Disziplinen noch immer maßgeblich (vgl. Terhart, 2009, S. 131-132). Im Gegensatz dazu liegt in der Pädagogik eher ein idiographisches Forschungsverständnis vor (vgl. Erpenbeck, 2012, S. 9). Aufgrund der fehlenden Gesetzmäßigkeiten im Software Engineering kann hier eher von einem idiographischem als einem nomothetisches Grundverständnis ausgegangen werden. Demzufolge sind die Herausforderungen im Software Engineering den Fragestellungen der Pädagogik näher als den Problemstellungen und Lösungsansätzen der anderen MINT-Fächer.

Gleichzeitig baut Software Engineering auf mathematisch-naturwissenschaftlichen Grundlagen auf und ist aus ihnen entstanden. Dies ist ein wesentlicher Grund dafür, dass die Vorgehensweisen, mit denen Software Engineering versucht, Herausforderungen zu begegnen, den ingenieursmäßigen Methoden anderer MINT-Fächer ähneln. Diese funktionieren jedoch aufgrund der verschiedenen Problemcharakteristika nicht richtig.

Während in den Human- und Sozialwissenschaften systematische, methodische Vorgehensweisen die Wissenschaftlichkeit der Ergebnisse sicherstellen, können Naturwissenschaften das methodische Vorgehen aufgrund der zugrundeliegenden Ursache-Wirkungszusammenhänge zugunsten einer Ergebnisorientierung vernachlässigen. Im Software Engineering fehlen jedoch genau diese Kausalketten, die man anwenden könnte, um zu tragfähigen Ergebnissen zu kommen.

Im Software Engineering führen aufgrund der Nähe zu Ingenieurs- und Naturwissenschaften und eines dort zugrundeliegenden nomothetischen Grundverständnisses diskursive Verfahren zu einem Konsens, der als wissenschaftliche Arbeitsgrundlage definiert wird, ohne auf einem zielorientierten methodischen Vorgehen aufzubauen oder dieses zumindest transparent darzulegen. Eine methodenorientierte, nachvollziehbare Vorgehensweise, wie sie etwa Gudjons für

die Pädagogik beschreibt (2006, S. 55), ist im Software Engineering bisher nicht üblich. Hier können Ansätze, die auf humanwissenschaftlichen Grundvorstellungen basieren, bessere Lösungsvorschläge liefern.

Allerdings zeigte die Pädagogik bisher kaum Interesse für das Themengebiet der MINT-Ausbildung an Hochschulen, geschweige denn für die Ausbildung im Software Engineering. Bislang stellen sich noch vorwiegend Lehrende des Software Engineering diesen Fragen, das Interesse auf pädagogischer Seite wächst erst langsam. Gerade hier wären jedoch entsprechendes Fachwissen und geeignete Forschungsmethoden vorhanden, die helfen können, die Fragen der Software-Ingenieure zu beantworten. Diese Verbindung soll diese Arbeit leisten.

Bisherige vereinzelte Publikationen zu kompetenzbezogenen oder fachdidaktischen Fragestellungen des Software Engineering konzentrieren sich entweder auf (einzelne) soziale Kompetenzen wie z.B. Teamarbeit (vgl. z.B. Perez Martinez, Garcia Martin & Sierra Alonso, 2014) oder auf Erfahrungsberichte zu einzelnen didaktischen Methoden (vgl. z.B. Hagel & Mottok, 2011). Derzeit fehlen eine systematische Forschung und ein umfassendes Bild zu fachdidaktischen Fragestellungen im Software Engineering.

Dies legt nahe, Überlegungen in Richtung einer Fachdidaktik für Software Engineering anzustellen.

3 Begründung und Aufbau des Forschungsdesigns

3.1 Rahmenbedingungen

Aus einem Arbeitskreis bayerischer Professoren entstand im Rahmen des Qualitätspakts Lehre ein Projekt mit dem Ziel, auf dem Gebiet des Lernens von Software Engineering zu forschen und dadurch Lehre von Software Engineering weiterzuentwickeln und auf wissenschaftlich fundierte Ergebnisse zu gründen. Vorteil ist neben der Motivation der Lehrenden, dass die Lehrveranstaltungen bereits existieren, stark anwendungsorientiert ausgerichtet und in Curricula und Studienpläne eingebettet sind. Dadurch ist es möglich, den Bezug zur praktischen Anwendung zu behalten und Veränderungen unmittelbar zu analysieren. Dadurch ist auch eine gewisse Nachhaltigkeit und Bedeutung dieser Forschung gegeben. Diese Rahmenbedingungen stellen eine nahezu optimale Basis dar, um im interdisziplinären Grenzbereich zwischen Software Engineering und Pädagogik zu forschen.

3.2 Zielsetzung

Ziel dieser Arbeit ist es, ein besseres Verständnis für die Herausforderungen des Lehrens und Lernens von Software Engineering an Hochschulen zu erlangen und auf dieser Grundlage passgenaue Lehrkonzepte zu entwickeln und zu evaluieren, um daraus eine Fachdidaktik für Software Engineering an Hochschulen abzuleiten. Dabei wird im Sinne der aktuellen Bologna-Diskussionen (vgl. *Empfehlungen zur Qualitätsverbesserung von Lehre und Studium,* 2008, S. 77-81) kompetenzorientiertes Studieren als Ziel dieser Fachdidaktik zugrunde gelegt, was bedeutet, dass Studierende nicht lediglich Fachwissen anhäufen, sondern vielmehr in die Lage versetzt werden sollen, dieses in Verbindung mit überfachlichen Kompetenzen situationsgerecht anzuwenden. Dieses komplexe Geflecht an Kompetenzen sollte die Hochschulausbildung im Software Engineering adressieren.

3.3 Theoretische Grundlagen und zugrundeliegender Kompetenzbegriff

Die Bologna-Reform rückt die bis dahin kaum beachtete Kompetenzorientierung in der Hochschullehre zunehmend in den Fokus. 2005 verabschiedete die KMK in Absprache mit dem BMBF einen Qualifikationsrahmen (vgl. Arbeitskreis Deutscher Qualifikationsrahmen (AK DQR), 2011), der Kompetenzen benennt, die Studierende nach Abschluss ihres Studiums haben sollen, was die Kompetenzorientierung fokussiert. Seitdem ist eine Diskussion entbrannt, was überhaupt Kompetenzen sind und vor allem wie sie gemessen werden können. So wurden Modelle entwickelt, die zunächst Kompetenzen strukturierten, etwa die Einordnung in personale, fachliche,

methodische und sozial-kommunikative Kompetenzarten (vgl. z. B. Gnahs, 2010, S. 26; Schneckenberg, 2008, S. 108), die etwa von Erpenbeck und Sauter (2013, S. 33) um aktivitäts- und umsetzungsorientierten Kompetenzen erweitert wurden. Kompetenzen wurden beschrieben, in Teilkompetenzen aufgegliedert wie etwa im Kompetenzatlas von Heyse und Erpenbeck (2010), und versucht zu definieren. Damit einher gingen Überlegungen zur Messung von Kompetenzen, was die Forschung schnell vor die Herausforderung der Operationalisierung stellt. "Selbst Ansätze und Studien, die einen engen und messbaren Kompetenzbegriff favorisieren (vgl. Klieme/Hartig 2007) schrecken offenbar vor der Komplexität von überfachlichen Kompetenzen zurück." (Bender, Lerch & Scheffel, S. 3)

Auch wenn sich „klassische“ Unterscheidungen wie diejenige in personale, fachlich-methodische und sozial-kommunikative Kompetenzen (vgl. etwa Erpenbeck & Sauter, 2013, S. 33) für dieses Forschungsvorhaben als zu allgemein erwiesen haben, so finden sich auch hier Kompetenzen aus diesen Bereichen, jedoch anders strukturiert (siehe Abschnitt 4.1.3).

Weinert definiert Kompetenzen als „die bei Individuen verfügbaren oder durch sie erlernbaren kognitiven Fähigkeiten und Fertigkeiten, um bestimmte Probleme zu lösen, sowie die damit verbundenen motivationalen, volitionalen und sozialen Bereitschaften und Fähigkeiten, um die Problemlösungen in variablen Situationen erfolgreich und verantwortungsvoll nutzen zu können“ (Weinert, 2001, S. 27-28). Auch dieser Kompetenzbegriff ist sehr allgemein. In der empirischen Bildungsforschung existieren Ansätze, Kompetenz als „erlernbare kontextspezifische kognitive Leistungsdispositionen“ (Hartig, 2008, S. 17) zu beschreiben. Diese Kompetenzdefinition berücksichtigt den fachlichen Kontext von Kompetenzen und wird daher häufig in den Fachdidaktiken als Grundlage herangezogen (vgl. Biehler & Leuders, 2014). Ein ähnliches Kompetenzverständnis liegt auch dieser Arbeit zugrunde.

Ferner wird hier davon ausgegangen, dass Kompetenzen trainierbar sind, jedoch mehr als reines Wissen erfordern und sich im Anwendungskontext einer komplexen Situation zeigen. Da diese Anwendungssituationen besonders im Software Engineering durch Interdisziplinarität charakterisiert sind und eine hohe Dynamik und Komplexität aufweisen, ist ein hohes Maß an interdisziplinären Kompetenzen erforderlich, um diese Situation zu meistern. Interdisziplinarität ist gekennzeichnet durch eine Vielzahl selbst wiederum komplexer Sozial- und Selbstkompetenzen (vgl. Bender, Lerch & Scheffel, S. 3).

Diese Arbeit legt den Fokus auf die Kompetenzen und die Lernprozesse von Studierenden, da diese im Sinne der Teilnehmerorientierung den Lernprozess selbst steuern (vgl. Siebert, 1997) und der Lehrende lediglich als „Lernermöglicher“ und Coach fungiert. Lernen im konstruktivistischen Grundverständnis geht davon aus, dass neues Wissen an bereits vorhandenes angeknüpft werden muss. Siebert (1999, S. 20) weist – ebenso wie bereits Rogers (1969) in seinen „Principles of Learning“ – zudem darauf hin, dass das zu Lernende eine Bedeutung für die Lernenden haben sollte und dies eine Voraussetzung für die Relevanz der Lernziele ist. Diesen Zugang zu finden und Studierende für das Thema Software Engineering zu öffnen, indem Problembewusstsein geschaffen wird, ist eine zentrale Herausforderung für die Lehre von Software Engineering. Ebenso müssen die für Software Engineering erforderlichen kontextspezifischen überfachlichen Kompetenzen in der Hochschullehre adressiert werden. Dabei steht der Lernende im Mittelpunkt der Überlegungen. Auch wenn die entsprechenden Lehrendenpersönlichkeiten bei didaktischen Überlegungen nicht außer Acht gelassen werden dürfen, so liegt auf ihnen nicht das zentrale Augenmerk in dieser Arbeit. Vielmehr fußen die Überlegungen zu einer Fachdidaktik Software Engineering auf der zentralen Frage, wie den Studierenden das Lernen, Verstehen und Anwenden dieses Faches ermöglicht und erleichtert werden kann.

3.4 Überblick über das Forschungsdesign

Die Entwicklung einer interdisziplinären Fachdidaktik für Software Engineering orientiert sich an drei zentralen iterativen Schritten:

1. Kompetenzziele: Um eine genauere Vorstellung zu erhalten, was das Ziel der Hochschulausbildung im Software Engineering darstellt, wurden die erforderlichen Kompetenzen im Software Engineering ermittelt und in einem Kompetenzprofil beschrieben (vgl. Sedelmaier & Landes, 2014d, 2015d / Anhang 1.6). Dabei wurde besonderes Augenmerk auf die überfachlichen Kompetenzen gelegt (Abschnitt 4.1).

2. Evaluation: Weiterhin ist es erforderlich, Lehrveranstaltungen im Software Engineering zu evaluieren, um schließlich wiederum daraus Rückschlüsse für deren Weiterentwicklung und die Entwicklung einer Fachdidaktik zu erhalten. Hierzu wurde ein Evaluationsinstrument entwickelt, das auf die Hochschulausbildung im Software Engineering zugeschnitten ist und Lehrveranstaltungen basierend auf einem multiperspektivischen Assessment studentischer Kompetenzen evaluiert (Abschnitt 4.2).

3. Didaktische Umsetzung: Die erforderlichen Kompetenzen sollen in der Hochschulausbildung gefördert werden. Dazu wurden vorhandene Lehrveranstaltungen analysiert und kompetenzorientiert weiterentwickelt. Dies mündete in einen Vorschlag für ein zielgerichtetes Vorgehen, das auf allgemeindidaktischen Theorien fußt, diese kombiniert und ergänzt (vgl. Sedelmaier & Landes, 2015a, 2015c / Anhang 1.5) (Abschnitt 4.3).

Dieses Forschungsdesign ist pädagogisch begründet und bedient sich im Wesentlichen aus zwei vorhandenen Leitkonzepten:

Besonders bei der Erforschung des Kompetenzprofils (Schritt 1) orientiert sich das Vorgehen an der Grounded Theory (vgl. Glaser & Strauss, 2010; Sedelmaier & Landes, 2013), um einer hermeneutischen Tradition folgend die Anforderungen an Software-Ingenieure zu verstehen. Das Forschungsdesign wurde zu Beginn des Vorhabens detailliert beschrieben (vgl. Sedelmaier & Landes, 2012).

Das Vorgehen bei der Evaluation (Schritt 2) kann der empirischen Bildungsforschung bzw. der Lehr-Lern-Forschung zugeordnet werden. Dabei wird ein umfassendes Verständnis von Kompetenz zugrunde gelegt, wie etwa in einer gängigen Definition nach Weinert (2001, S. 27-28) oder auch in der empirischen Bildungsforschung nach Hartig (2008, S. 17). In die Evaluation der Lehre fließen fundierte Forschungen der Berufsbildung ein. Rauner (2011) misst berufliche Kompetenzen und entwickelte dazu u.a. ein methodisches Vorgehen. Dieses wurde auf Software Engineering angepasst und erweitert, um die Qualität der Hochschullehre anhand des Kompetenzzuwachses von Studierenden zu bewerten (vgl. Sedelmaier & Landes, 2014b / Anhang 1.2).

Die Schritte 1 und 2 führen zu einem theoretisch fundierten Kompetenzmodell, das basierend auf pädagogischen Theorien den Rahmen für das Forschungsdesign zur Entwicklung einer interdisziplinären Fachdidaktik Software Engineering spannt. Dieses Kompetenzmodell fußt im Wesentlichen auf einer dreigliedrigen Struktur aus fachlichen, allgemeinen sowie kontextsensitiven überfachlichen Kompetenzen (vgl. Sedelmaier & Landes, 2015d / Anhang 1.6), die sich auf drei Niveaustufen ansiedeln lassen.

Das Vorgehen bei der Entwicklung von Lehr-Lern-Konzepten und das Kompetenzmodell basieren auf pädagogischer Theorie und werden unmittelbar in der Hochschullehre angewandt (Schritt 3, Abschnitt 4.3).

Das dreigliedrige Forschungsdesign sowie das zugrundeliegende theoretische Modell ebnen den Weg zu einer Hochschul-Fachdidaktik für Software Engineering (vgl. Sedelmaier & Landes, 2015c / Anhang 1.5). Die einzelnen Schritte werden im Folgenden näher erläutert.

4 Umsetzung des Forschungsdesigns und wesentliche Ergebnisse

4.1 Software Engineering Body of Skills (SWEBOS)

Zunächst waren die Ziele der Software Engineering-Ausbildung zu klären. Dazu sind eine Vielzahl von Fragen zu beantworten, etwa: Welche überfachlichen und fachlichen Kompetenzen benötigt ein Software-Ingenieur und wie beeinflussen sich diese Kompetenzen gegenseitig? Was genau ist mit bestimmten Kompetenzen gemeint? Was bedeutet etwa kommunikative Kompetenz im Kontext von Software Engineering? Was genau ist mit Begriffen wie „Teamarbeit“ oder „Kommunikative Kompetenz“ im Kontext von Software Engineering gemeint? Wie prägen sie sich im speziellen Kontext von Software Engineering aus? Soll ein Software Entwickler drei verschiedene Sprachen beherrschen? Oder soll er in der Lage sein, ein Anforderungsdokument zu schreiben? Oder ist es vielmehr wichtig, dass er in der Lage ist, kommunikative Methoden und Techniken anzuwenden, die es ihm ermöglichen, Anforderungen zu erheben, zu präzisieren und zu dokumentieren? Benötigt er dafür nicht auch die Fähigkeit, interdisziplinär zu kommunizieren?

Da die fachliche Expertise relativ unstrittig ist, lag der Fokus auf überfachlichen Kompetenzen. Um sozialwissenschaftlich fundierte Ergebnisse und ein tiefgehendes Verständnis der erforderlichen Kompetenzen zu erhalten, wurde ein Vorgehen gewählt, das der Grounded Theory (vgl. Glaser & Strauss, 2010) folgt. Ergebnis ist das Kompetenzprofil „Software Engineering Body of Skills“ (SWEBOS) für Software Engineering mit besonderer Berücksichtigung kontextsensitiver überfachlicher Kompetenzen (Sedelmaier & Landes, 2015d / Anhang 1.6). Das entstandene Kompetenzprofil SWEBOS dient als Ziel und Maßstab für die Hochschulausbildung, an dem sich die Lehre orientiert. Vorgehen, Methodik und Ergebnisse wurden in Sedelmaier und Landes (2015d) (Anhang 1.6) bereits veröffentlicht. Daher erfolgt an dieser Stelle eine Zusammenfassung der wesentlichen Aspekte.

4.1.1 Stand der Forschung und vorhandene Konzepte

Im Software Engineering sind in letzter Zeit Publikationen zu finden, die sich damit befassen, was einen “guten” Software-Ingenieur auszeichnet (vgl. z. B. Li, Ko & Zhu, 2015). Meist fehlen jedoch – wie bereits erläutert – die methodischen Vorgehensweisen, um systematisch zu validen Ergebnissen zu kommen, die über Erfahrungsberichte oder Meinungsäußerungen hinausgehen, wie etwa bei Peters und Moreno (2015).

Hinweise auf mögliches Fachwissen eines Software-Ingenieurs liefert der „Guide to the Software Engineering Body of Knowledge“ (SWEBOK), der mittlerweile in der dritten Überarbeitung erschienen ist (vgl. Bourque & Fairley, 2014). Dieser ist im Software Engineering als Standard anerkannt und beinhaltet verschiedene Wissensgebiete des Software Engineering. Mittlerweile wird er ergänzt um ein „Software Engineering Competency Model“ (SWECOM) (vgl. IEEE Computer Society, 2014).

Auch wenn in der aktuellen Version von SWEBOK einzelne überfachliche Kompetenzen genannt werden, reichen die Richtlinien SWEBOK und SWECOM alleine nicht aus, um ein umfassendes Bild über die erforderlichen Kompetenzen im Software Engineering zu erhalten. Dafür gibt es mehrere Gründe:

Zum einen ist auf der methodischen Seite nicht nachvollziehbar, woher die Daten und Wissensgebiete in SWEBOK und SWECOM kommen. Es ist kein nachvollziehbares Forschungsdesign beschrieben, das für die Gültigkeit der Daten sprechen würde. Jede sozialwissenschaftliche Untermauerung fehlt.

Inhaltlich handelt es sich zum anderen bei den Wissensgebieten in SWEBOK zunächst um eine reine Themensammlung, in SWECOM sind lediglich Aufgaben beschrieben. Zwar sind in SWEBOK die Themenfelder mit der Lernzieltaxonomie nach Bloom (1972) und in SWECOM die Aufgaben mit Niveaustufen hinterlegt, jedoch liegt der Schwerpunkt im SWEBOK auf fachlichem Wissen und in SWECOM auf Tätigkeiten. Bei SWEBOK handelt es sich eher um eine fachliche Inhaltssammlung, die die Disziplin Software Engineering eingrenzt. Größtes Manko hier sind jedoch die fehlenden überfachlichen Kompetenzen eines Software-Ingenieurs. Die wenigen genannten überfachlichen Kompetenzen im SWEBOK werden zudem lediglich mit Schlagworten benannt, ohne nähere Definition, was damit genau gemeint sein könnte.

Weder SWEBOK noch SWECOM enthalten operationalisierte Kompetenzen, noch bieten sie sinnvolle Zusammenhänge zwischen den Themen oder Aufgaben an.

Ähnlich verhält es sich mit den ACM Computer Science Curriculum (vgl. IEEE Computer Society & Association for Computing Machinery ACM, 2004): Sofern überhaupt überfachliche Kompetenzen genannt werden, bewegen sie sich auf einem sehr allgemeinen Niveau und werden nicht kompetenzorientiert beschrieben. Erschwerend kommt hinzu, dass Software Engineering lediglich ein kleiner Teilbereich des ACM Curriculum ist.

Inhaltlich lässt sich also feststellen, dass Kompetenzen, die bottom up die beruflichen Herausforderungen im Software Engineering umfassend und zunächst fachspezifisch verstehen, bisher nicht bekannt sind. Hinzu kommt das Fehlen überfachlicher Kompetenzen. Ferner wird keines dieser Konzepte den in den Sozialwissenschaften

üblichen methodischen Standards gerecht. Ein Stand der Forschung ist also nicht zu erkennen. Diesen aufgezeigten Defiziten soll mit dem SWEBOS Rechnung getragen werden.

4.1.2 Vorgehensweise

Um die erforderlichen Kompetenzen im Software Engineering zu erforschen, wurde ein qualitativer, datengetriebener Forschungsansatz zur Theoriebildung gewählt, der auf der Grounded Theory (vgl. Glaser & Strauss, 2010) basiert.

Ziel war es, Kompetenzen eines Software-Ingenieurs bottom up zu verstehen und zu erklären, nicht nur ex kathedra zu definieren. In dieser Arbeit wurde den Leitideen der Grounded Theory folgend die Theorie generiert, nicht „nur“ verifiziert (vgl. Glaser & Strauss, 2010, S. 49).

Dazu wurde die in Abb. 1 skizzierte qualitative Vorgehensweise gewählt.

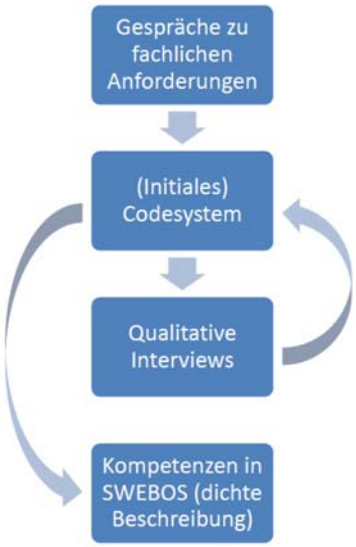


Abbildung 1 Qualitatives Vorgehen

Vorhandenes Ausgangsmaterial wurde in einem qualitativen Verfahren initial ausgewertet und codiert, so dass ein erstes Codesystem entstand. Daraus wurde ein Interviewleitfaden entwickelt, Interviews mit Praktikern aus dem Software Engineering geführt und so über den Einblick in den Berufsalltag von Software-Ingenieuren ein tiefgehendes Verständnis für die erforderlichen überfachlichen Kompetenzen im Software Engineering generiert.

Das Codesystem stellt einen Abstraktionsschritt dar und wurde bottom up aus den Daten heraus entwickelt. Es dient der Strukturierung der sogenannten dichten Beschreibung (vgl. Geertz, 1987), die dazu dient, die Realität des Software

Engineering und insbesondere die dort erforderlichen kontextsensitiven überfachlichen Kompetenzen zu verstehen und in einer sehr reichhaltigen und wenig abstrakten Darstellung, aber trotzdem allgemeingültig zu charakterisieren (vgl. Mills, 2010, S. 942). Das Codesystem ist quasi ein Zwischenschritt zur „Ordnung“ der Zusammenhänge, die dann wiederum in der dichten Beschreibung abgebildet werden.

Das Vorgehen ist in Sedelmaier und Landes (2015d) (Anhang 1.6) im Detail erläutert.

4.1.3 Ergebnisse

Das entstandene Kompetenzprofil für Software Engineering, der „Software Engineering Body of Skills“ (SWEBOS) (Sedelmaier & Landes, 2015d / Anhang 1.6), ergänzt und vernetzt das SWEBOK und SWECOM mit überfachlichen Kompetenzen. SWEBOS stellt ein überfachliches Soll-Kompetenzprofil für Software Engineering nach Abschluss des Studiums dar.

Wesentliche Ergebnisse in diesem Forschungsabschnitt sind das Kompetenzverständnis, Erkenntnisse über eine geeignete Darstellungsform sowie die inhaltlichen Ergebnisse an sich.

Kompetenzverständnis

Die Ergebnisse aus den Interviews führten zu einer Verfeinerung des Codesystems. Dabei stellte sich zur besseren Strukturierung eine Unterscheidung in intra- und inter-personelle Kompetenzen als sinnvoll heraus. Inter-personelle Kompetenzen beziehen sich auf Kompetenzen, die im Umgang und der Zusammenarbeit mit anderen Menschen besonders zum Tragen kommen, wie beispielsweise kommunikative Kompetenzen. Dagegen liegen intra-personelle Kompetenzen in der Person und ihren Einstellungen und Werten selbst.

Des Weiteren hat sich eine Unterscheidung in fachliche Kompetenzen sowie kontextsensitive und allgemeine überfachliche Kompetenzen als sinnvoll erwiesen. Diese Unterscheidung ist notwendig, um den Fachbezug zum Software Engineering herzustellen und ein tiefes Verständnis für die Bedeutung überfachlicher Kompetenzen im Kontext von Software Engineering zu erreichen. Die Unterscheidung in fachliche, kontextsensitive überfachliche und allgemeine überfachliche Kompetenzen stellt das zugrundeliegende pädagogische Kompetenzverständnis dar.

Fachliche Kompetenzen beziehen sich auf das Fachwissen eines Software-Ingenieurs, etwa das Beherrschen einer Programmiersprache oder die Kenntnis von Vorgehensmodellen. Hier lag jahrelang der Schwerpunkt der Hochschulausbildung, auch aus der Vorstellung heraus, dass sich die überfachlichen Kompetenzen von selbst einstellen. Fachwissen alleine reicht jedoch nicht aus, um als Software-Ingenieur tätig zu werden. Wie in jedem Beruf muss es um überfachliche Kompetenzen ergänzt werden, die während des Studiums auch berücksichtigt werden müssen.

Wie erwähnt lassen sich bei überfachlichen Kompetenzen allgemeine überfachliche und kontextsensitive überfachliche Kompetenzen unterscheiden. Allgemeine überfachliche Kompetenzen können losgelöst vom fachlichen Kontext trainiert werden (z. B. Präsentationstechniken), während kontextsensitive überfachliche Kompetenzen sehr stark vom Fachlichen einer Disziplin, in diesem Fall des Software Engineering, charakterisiert werden. Kontextsensitive überfachliche Kompetenzen sollten im Zusammenhang mit den fachlichen Kompetenzen im Kontext von Software Engineering trainiert werden. Sie haben eine eigene Bedeutung im Kontext von Software Engineering, wie z. B. kommunikative Fähigkeiten oder Kompetenzen, in einem Team zu arbeiten.

Die Fähigkeit, sich auf die Welt des jeweiligen Kunden oder Gesprächspartners einzulassen, ist eine Kernkompetenz im Software Engineering und stellt eine solche kontextsensitive überfachliche Kompetenz dar. Ein Verständnis dafür, wie diese Kompetenz genau charakterisiert ist und wie ein erfolgreicher Software-Ingenieur agiert, der sie besitzt, ist ein inhaltliches Ergebnis dieses Forschungsschrittes.

Einzeln gesehen erscheinen kontextsensitive überfachliche Kompetenzen auf den ersten Blick wenig „besonders“, bekommen jedoch bei näherer Betrachtung im Kontext der jeweiligen Fachlichkeit eine spezifische Charakteristik. Zudem bilden sie zusammen genommen in Verbindung mit fachlichen und allgemeinen überfachlichen Kompetenzen ein komplexes Geflecht, somit ein spezielles Kompetenzprofil für Software Engineering.

So erlernen Informatik-Studierende als fachliche Kompetenz zwar, wie man zum Beispiel einen Prozess modelliert, jedoch benötigen sie nicht nur die Notation, sondern zusätzlich Methoden und Techniken überfachlicher Natur, die es ihnen ermöglichen, die zu modellierenden Informationen überhaupt erst zu bekommen. Dazu gehören zum Beispiel an den Kontext von Software Engineering angepasste, zielgerichtete Fragetechniken und die systematische Vorbereitung auf ein Kundengespräch genauso wie die Moderation dieses Kundengesprächs. Fragetechniken im Bereich des Software Engineering sind wesentlich durch den Kontext bestimmt, hier also die Gewinnung von Informationen zu Prozessen innerhalb einer Organisation und die dahinterliegende

Modellvorstellung, was zur Beschreibung von Prozessen gehört. Fragen zielen also etwa auf die Ermittlung von Aktivitäten und die Ereignisse, die sie auslösen, um Anforderungen an ein Softwaresystem zu erfassen. Die erfragten Informationen und somit auch die Fragetechniken sind darauf ausgerichtet, Informationen zu erhalten, die für die Entwicklung von Software wesentlich sind. Ein zentrales Ziel von Fragetechniken im Software Engineering ist zudem die Erhebung impliziter Anforderungen:

"Anstatt dem anderen aber die eigene handlungsleitende 'Logik' zu enthüllen, vermittelt der Experte/Kunde zumeist nur Beispiele aus der Praxis oder seine subjektive Auswahl des Arbeitsprozesses und eventuell der Arbeitsumgebung. Selten nur ist er in der Lage, Prinzipien, nach denen er handelt, in Worte zu fassen. Die Entwickler müssen ihm also helfen, die impliziten Regeln seines Tuns zu erkennen bzw. das zu strukturieren, was er weiß." (Funken, 2001, S. 138)

Daher erhalten allgemein scheinende kommunikative Kompetenzen wie etwa Fragetechniken im Kontext von Software Engineering eine spezielle Ausprägung und unterscheiden sich von Fragetechniken etwa in der Sozialen Arbeit. So sind etwa Fragetechniken auf den ersten Blick keine besondere kontext-sensitive überfachliche Kompetenz. Es macht jedoch wenig Sinn, Informatikstudierende in Fragetechniken auszubilden, wenn der Anwendungsbezug, wie etwa der Einsatz in einem Kundengespräch zur Anforderungserhebung, fehlt. Fragetechniken erhalten eine besondere Ausprägung durch das Ziel, Anforderungen an ein Softwaresystem zu erheben.

Zudem sind diese Kompetenzen mit weiteren fachlichen und überfachlichen Kompetenzen verknüpft, weil eine Fachkompetenz wie etwa eine Modellierungstechnik für Geschäftsprozesse die Art der zu erfragenden Information beeinflusst und der Frageprozess mittels Arbeitstechniken strukturiert werden muss. Hinzu kommt also neben dem Kontext die Kombination mit weiteren erforderlichen überfachlichen kontextsensitiven Kompetenzen wie etwa Arbeitstechniken. Eben genau diese Kompetenzen sind unabdingbar, damit technisches Fachwissen wie Modellierungsnotationen überhaupt erst anwendbar wird. Wenn ein Informatiker nicht in der Lage ist, die notwendigen Informationen vom Kunden zu erfragen, gibt es auch nichts, was als Prozessmodell dargestellt werden kann und woraus er Anforderungen ableiten könnte.

Insgesamt stellt Software Engineering spezielle Anforderungen an die Entwickler, um z.B. Anforderungen zu erheben. Wie in einer anderen Fachdisziplin auch kommunikative Kompetenz gefordert ist, so müssen auch Software-Ingenieure die Brücke zwischen den verschiedenen Welten schlagen und die Inhalte in ihre eigene

Informatiker-Fachwelt „übersetzen“. Dies kann zudem individuell auf sehr unterschiedliche Art und Weise erfolgen und jeder Informatiker bringt seinen eigenen Stil und seine Persönlichkeit in den Arbeitsprozess ein:

"Die relative Vielfalt der Software-Entwicklungsmodelle, die zahlreichen Programmiersprachen und -tools und der 'universelle' Charakter des Computers ermöglichen im Prinzip eine weitgehende Ausdifferenzierung seiner Gebrauchsweisen. Software-Entwicklung wird gleichsam zum Vehikel der Stilbildung und Individuation; Ausbildung, Wissen, Talent, Persönlichkeit und organisatorische Bedingungen gerinnen zu einem je unverwechselbaren Entwicklungsstil." (Funken, 2001, S. 61)

Darstellungsform

Die Darstellungsform von SWEBOS resultiert aus dem interdisziplinären und anwendungsbezogenen Kontext, in dem die vorliegende Arbeit angesiedelt ist. Zum einen soll das Kompetenzprofil SWEBOS sozialwissenschaftlichen Standards genügen und auch für die Pädagogik und verwandte Disziplinen einen Nutzen bringen. Zum anderen soll SWEBOS auch für technisch geprägte Fachdisziplinen als Kompass in der Hochschulausbildung dienen können und so eine praktische Verwendung erfahren. Die Darstellungsweise in Tabellenform wurde bewusst ausgewählt, um allen beteiligten Fachdisziplinen in diesem interdisziplinären Kontext gerecht zu werden. Während Sozial- und Humanwissenschaften sich vornehmlich mittels Prosatexten und sehr einfachen informellen Skizzen ausdrücken, sind technisch geprägte Disziplinen, zu denen die Informatik und somit auch Software Engineering gehört, von Beginn der Ausbildung an darauf trainiert, sehr komplexe Sachverhalte zu abstrahieren und mittels bildlicher, semantisch definierter Darstellungsformen zu modellieren und zu strukturieren. Während pädagogisch geprägte Wissenschaftler häufig nach Erläuterungen und Erklärungen in Prosaform fragen, bitten Informatiker und Techniker meist um eine bildliche Modellierung des Sachverhalts. Um beiden Disziplinen Rechnung zu tragen und um SWEBOS für beide zugänglich zu machen, wurde eine strukturierte Beschreibung in Tabellenform in Verbindung mit einer detaillierten, dichten Beschreibung als Darstellungsweise ausgewählt (vgl. Sedelmaier & Landes, 2015d / Anhang 1.6): Auf oberster Stufe wird eine kurze Beschreibung der Kompetenz angegeben. Auf zweiter Ebene befindet sich eine detailliertere, „dichte“ Beschreibung (vgl. Mills, 2010, S. 942), was unter dieser Kompetenz zu verstehen ist. Dritte Ebene bilden exemplarische Handlungsanker, wie sich jemand, der diese Kompetenz besitzt, verhalten könnte.

Inhalte

Die in SWEBOS enthaltenen kontextsensitiven überfachlichen Kompetenzen lassen sich wie folgt zusammenfassen, auch wenn das hermeneutische Verständnis und damit der Kontext erst in Verbindung mit den zugeordneten Erläuterungen und dichten Beschreibungen ersichtlich werden:

- Kompetenzen für die professionelle Zusammenarbeit mit anderen Menschen (Z)
- Kommunikative Kompetenzen (K)
- Kompetenzen, um die eigene Arbeit zu strukturieren (S)
- Personale Kompetenzen (P)
- Fähigkeit, komplexe Vorgänge und Systeme sowie Zusammenhänge zu verstehen (Problembewusstsein) (V)
- Fähigkeit, das eigene Wissen und Können, die eigenen Kompetenzen auf konkrete, neue Situationen flexibel und kreativ anzuwenden (Lösungskompetenz) (L)

4.1.4 Zusammenfassung, Bewertung und Ausblick

Mit SWEBOS (vgl. Sedelmaier & Landes, 2015d / Anhang 1.6) liegt ein Kompetenzprofil vor, das beschreibt, welche überfachlichen kontextsensitiven Kompetenzen im Software Engineering in Verbindung mit den fachlichen Kompetenzen erforderlich sind. Zentrale Resultate sind das aus dem Forschungsprozess entstandene Kompetenzverständnis und die identifizierten überfachlichen Kompetenzen im Software Engineering. Beides wurde in verschiedenen Lehrveranstaltungen unterschiedlicher Studiengänge bei der Lehrveranstaltungsplanung mehrfach praktisch angewandt (siehe Abschnitt 4.3), indem etwa Lehrziele aus dem Kompetenzprofil SWEBOS abgeleitet und operationalisiert wurden. Des Weiteren orientiert sich die Kompetenzbewertung inhaltlich sehr stark an SWEBOS und folgt auch demselben Kompetenzverständnis. In Summe legen diese Aspekte den Schluss nahe, dass SWEBOS auch längerfristig tragfähig ist. Dennoch ist eine iterative Überprüfung und Weiterentwicklung von SWEBOS wünschenswert, um etwa aufgrund der Forschungsergebnisse in Verbindung mit dem Evaluationsinstrument SECAT die Struktur auf inhaltlicher Ebene noch weiter zu verfeinern und aufzugliedern. Die einzelnen Kompetenzbereiche werden sich in einer zweiten Version möglicherweise stärker voneinander abgrenzen lassen. Das geschilderte Vorgehen ist iterativ: SWEBOS und SECAT beeinflussen sich im Sinne

der Grounded Theory immer wieder in einem iterativen Entwicklungsprozess wechselseitig.

4.2 Bewertungsansatz Software Engineering Competence Assessment Tool (SECAT)

Nachdem ein Kompetenzprofil für Software-Ingenieure vorliegt, aus dem Lehr-Lernziele entwickelt werden, stellt sich die nächste Frage im Forschungsprozess zur Entwicklung einer Fachdidaktik (vgl. Plöger, 1999, S. 17), nämlich wie Lehre gestaltet werden kann, damit möglichst viele Lehr-Lernziele erreicht werden können. Im Software Engineering beinhalten solche Lehr-Lernziele meist überfachliche Kompetenzen.

4.2.1 Stand der Forschung

In der Bildungsforschung wurden zahlreiche quantitative large-scale Studien zur Kompetenzmessung aufgesetzt und die nationale Perspektive mit anderen verglichen: PISA (vgl. OECD, 2013), Nationales Bildungspanel (NEPS) (vgl. Artelt, Weinert & Carstensen, 2013; Blossfeld, Roßbach & Maurice, 2011), Programme for the International Assessment of Adult Competencies (PIAAC) (vgl. Rammstedt & Ackermann, 2013), um nur einige Wesentliche zu nennen.

Häufig werden nicht zuletzt aufgrund großer Stichproben und Geltungsbereiche sehr allgemeine Begriffe oder sehr allgemeingültige Kompetenzen als Forschungsgegenstand herangezogen, gerade auch wenn überfachliche Kompetenzen im Spiel sind. So erfasst PIAAC ähnlich der PISA-Studie grundlegende Kompetenzen – im Gegensatz zu PISA allerdings von Erwachsenen. Grundlegende Kompetenzen in PIAAC sind Lesekompetenz, alltagsmathematische Kompetenz und technologiebasiertes Problemlösen. "Nach der allgemeinen Definition umfasst technologiebasiertes Problemlösen in PIAAC die Verwendung von digitalen Technologien, Kommunikationswerkzeugen und Netzwerken mit dem Ziel, Informationen zu beschaffen und zu bewerten, mit anderen zu kommunizieren sowie alltagsbezogene Aufgaben zu bewältigen." (Rammstedt & Ackermann, 2013, S. 62) Auch diese auf den ersten Blick etwas spezifischer anmutende Kompetenz ist bei genauerem Hinsehen eine Basiskompetenz auf recht allgemeinem Niveau.

Ähnliches ist bei den schulbezogenen Studien festzuhalten: "Entsprechend konzentriert sich die schulbezogene Forschung auf die Erhebung von grundlegenden Fähigkeiten, wie die Third International Mathematics and Science Study (TIMSS), die mathematische und naturwissenschaftliche Fähigkeiten erfasst, oder das Programme for International Student Assessment (PISA) mit der Erfassung von mathematischen

und naturwissenschaftlichen Kompetenzen sowie der Lesekompetenz. Diese Kompetenzen werden in standardisierten Leistungstest gemessen. Dazu werden Aufgaben konzipiert, die eine 'richtige' oder 'falsche' Lösung zulassen. Beispielsweise kann die mathematische Fähigkeit dadurch überprüft werden, dass die Befragten Rechenaufgaben lösen. Unterschiedliche Leistungsniveaus können so entlang festgelegter Kriterien differenziert werden." (Braun & Hannover, 2008, S. 155)

Nicht nur, dass diese Studien sehr stark auf Basiskompetenzen – meist im Schulbereich – fokussieren (vgl. Braun & Hannover, 2008, S. 155), fehlt diesen Studien neben der Prüfung von Handlungskompetenzen auch der Bezug zu Software Engineering und dem spezifischen Verständnis, was mit diesen Kompetenzen genau gemeint ist, so wie es in SWEBOS beschrieben wurde.

Auch wenn es in der Schulforschung seit langem Ansätze zur Leistungsbewertung gibt (vgl. z. B. Jürgens, 1992), so fehlt ihnen die Kompetenzorientierung und sie fokussieren stattdessen stärker auf Wissenserwerb. Also sind vergleichende Studien wie etwa PISA (vgl. OECD, 2013) oder TIMSS (Trends in International Mathematics and Science Study) (vgl. Mullis, Martin, Foy & Arora, 2012) nicht auf den kompetenzorientierten Hochschulbereich übertragbar.

Das einer Bewertung zugrunde liegende Kompetenzverständnis, das etwa in der Psychologie ein anderes ist als in der Erziehungswissenschaft, beeinflusst das methodische Vorgehen bei der "Messung" oder Bewertung von Kompetenzen. Psychometrische Verfahren, wie sie vor allem in der Psychologie Anwendung finden, versuchen Kompetenzen zu quantifizieren. Dabei fokussieren sie jedoch häufig starr auf kognitive Leistungsdimensionen, aber eben nicht auf Kompetenzen (vgl. Bender, Lerch & Scheffel, S. 2). Hinzu kommt, dass Kompetenzen nicht einfach durch schriftliche Leistungstests abprüfbar sind. Auch durch psychometrische Tests oder psychologische Vorgehensweisen sind lediglich eingrenzbare methodische Kompetenzen, nicht aber überfachliche oder mehrfach komplexe Kompetenzen wie etwa interdisziplinäre Kompetenzen messbar (vgl. Bender, Lerch & Scheffel, S. 3).

Mit der zunehmenden Kompetenzorientierung durch die Bologna-Reform rücken auch überfachliche Kompetenzen und deren Messung in den Fokus des Forschungsinteresses (vgl. Barre, 2012, S. 104). Für eingrenzbare methodische Kompetenzen sind erste valide Bewertungsansätze entwickelt und einsatzbereit, wie etwa das „Berliner Evaluationsinstrument für selbsteingeschätzte studentische Kompetenzen“ (BEvaKomp) (Braun, 2008). Allerdings zielen vorhandene Kompetenzdiagnoseinstrumente entweder auf häufig erhobene und damit sehr allgemeine Kompetenzen oder lediglich auf Teilkomponenten einer Kompetenz (vgl.

Gnahs, 2010, S. 68). Insgesamt ist festzustellen, dass für komplexe überfachliche Kompetenzen noch tragfähige Ansätze fehlen.

Auch in der beruflichen Bildung sind international erprobte und validierte sowie empirisch und theoretisch fundierte Messverfahren entwickelt. So unterstützt etwa das Messinstrument von Rauner (2011) die Bewertung von Kompetenzen anhand komplexer Aufgabenstellungen in Gesellenprüfungen. Diese Verfahren konzentrieren sich jedoch noch auf den gewerblich-technischen Bereich und sind somit nicht direkt auf die kompetenzorientierte Hochschulausbildung übertragbar.

Zudem existieren seit der Einführung des Deutschen Qualifikationsrahmens (DQR) (Arbeitskreis Deutscher Qualifikationsrahmen (AK DQR), 2011) Forschungsansätze, die im Gegensatz zum Input den Bildungsprozess selbst und seinen Output- bzw. Outcome in den Mittelpunkt des Interesses stellen (vgl. Dehnbostel, Neß & Overwien, 2009, S. 48). Andere fokussieren auf die Perspektive des Lehrenden (vgl. z. B. Blömeke, Kaiser & Lehmann, 2011).

Aufgrund der Kontextbezogenheit des Kompetenzverständnisses und der fehlenden fachdidaktischen Forschungen zu Software Engineering existiert auch kein Ansatz zur Evaluation. Dieser wurde im Zuge dieser Arbeit entwickelt.

4.2.2 Ausgangslage

Da Untersuchungen zur Wirksamkeitsforschung didaktisch-methodischer Konzepte generell selten sind (vgl. Siebert, 2006, S. 14), wurde ein Evaluationsinstrument für Lehrkonzepte im Software Engineering entwickelt, um Erfolgsfaktoren des Lehrens und Lernens von Software Engineering zu identifizieren (vgl. Sedelmaier & Landes, 2014b / Anhang 1.2). Die Bewertung studentischer Kompetenzen liegt dabei als Gradmesser zugrunde und erfolgt mittels SECAT.

SECAT als Ansatz zur Kompetenzmessung bei Studierenden und die darauf aufbauende Analyse von didaktischen Konzepten basieren auf dem bereits erläuterten Kompetenzbegriff (Abschnitt 3.3) sowie dem darauf aufbauenden, bereits in Abschnitt 4.1 erläuterten Kompetenzverständnis aus SWEBOS. Die in SWEBOS zu Tage getretene Unterscheidung in allgemeine und kontextsensitive überfachliche sowie fachliche Kompetenzen spiegelt sich in SECAT ebenso wider wie die Differenzierung in personale und interpersonelle Kompetenzen, die ebenfalls aus dem Forschungsabschnitt SWEBOS resultiert. Diese Begriffe mit ihren Relationen bilden die Grundlage für das in SECAT entwickelte Kompetenzmodell. In SECAT wird also ebenfalls ein eigenes Verständnis von Kompetenz zugrunde gelegt, das auf die

spezifischen Eigenschaften der Domäne Software Engineering Bezug nimmt und in Abschnitt 4.1 beschrieben wurde.

4.2.3 Ziele und Intentionen von SECAT

Kompetenzen im Software Engineering sind sehr komplex. Um Lernprozesse im Software Engineering besser verstehen und Lehre kompetenzorientiert weiterentwickeln zu können, soll der Kompetenzzuwachs infolge eines Lehrkonzepts analysiert werden. Setzt man dann den tatsächlichen Kompetenzzuwachs in Relation zu den angewandten didaktischen Konzepten, kann so ein besseres Verständnis der Lehr-Lern-Zusammenhänge und eine gezielte Weiterentwicklung der Lehrkonzepte ermöglicht werden. Folglich müssen sich die zu erreichenden Kompetenzen im Evaluationsinstrument widerspiegeln.

Funktionales Fachwissen kann relativ einfach über klassische Prüfungsdesigns abgeprüft werden. Allgemeine überfachliche Kompetenzen können in unterschiedlicher Ausprägung in nahezu jeder Lehrveranstaltung in jedem Fach trainiert werden. Ein mögliches Evaluationsinstrument für diese Art von Kompetenzen ist etwa BEvaKomp (vgl. Braun, 2008).

Ein Evaluationsinstrument für eine Fachdidaktik des Software Engineering muss aber im Gegensatz dazu die Besonderheiten des Faches berücksichtigen und darauf abgestimmt sein. Ein Messinstrument der Fachdidaktik des Software Engineering sollte neben den fachlichen Kenntnissen besonders die kontextsensitiven überfachlichen Kompetenzen einschließlich der fachspezifischen Besonderheiten berücksichtigen. Gradmesser in SECAT sind demnach besonders die fachlichen und kontextsensitiven überfachlichen Kompetenzen, die Studierende aufgrund der Lehrveranstaltung weiterentwickelt haben. Diese gilt es zu evaluieren, um daraus Rückschlüsse auf die Lehrkonzepte zu ziehen. Dazu ist ein empirisch fundiertes Evaluationsinstrument erforderlich, das den Regeln wissenschaftlicher Evaluation folgt (vgl. z.B. Reischmann, 2003; Rindermann, 2003).

Das Evaluationsinstrument SECAT soll somit folgenden Anforderungen genügen:

- Ein Messinstrument sollte nicht lediglich nur Momentaufnahmen aus spezifischen Lehrveranstaltungen aufzeigen, sondern zusätzlich auch sämtliche Kompetenzen des Software Engineering in ihrer gesamten Komplexität abbilden können. Eine einzige Lehrveranstaltung reicht nicht aus, um alle erforderlichen Kompetenzen auszubilden. Dafür sind mehrere aufeinander abgestimmte Lehrveranstaltungen notwendig, die sich ergänzen und Kompetenzen Schritt für Schritt trainieren. Zum einen scheint es sinnvoll, die

Kompetenzentwicklung durch sowie die Wirksamkeit von einzelnen Lehrveranstaltungen oder didaktischen Methoden zu evaluieren. Zum anderen soll die entwickelte Evaluationsmethodik auch als summative Evaluation auf die Hochschulausbildung im Software Engineering als Gesamtes blicken können. Zudem bringen die Studierenden individuelles Vorwissen mit, das zu verschiedenen Zeitpunkten im Studium unterschiedlich ist. Es ist also erforderlich, Kompetenzen auf verschiedenen Niveaustufen zu evaluieren. Dann kann SECAT während des gesamten Studiums eingesetzt werden und sowohl die individuellen Kompetenzen einzelner Studierender messen, als auch die Lehre adressatenorientiert evaluieren und weiterentwickeln. Dies wird durch das zugrunde liegende Kompetenzmodell möglich.

- Ziel ist es zudem, auch kontextsensitive, für Software Engineering spezifische Kompetenzen zu messen. Es ist nicht ausreichend, ausschließlich allgemeine überfachliche Kompetenzen zu bewerten wie etwa BEvaKomp (vgl. Braun, 2008).
- Gutes Software Engineering zeigt sich sowohl in den Abläufen und Prozessen während der Entwicklung, als auch im Ergebnis, der fertigen Software. Daher müssen auch beide Aspekte in die Bewertung der Lehre mit einfließen: Befähigt sie die Studierenden dazu, mit Hilfe eines systematischen Arbeitsprozesses zu guten Ergebnissen zu kommen? Eine Bewertung rein über Prozesse greift zu kurz, denn auch zielgerichtete Arbeitsprozesse führen im Software Engineering nicht zwangsläufig zu perfekten Ergebnissen. Umgekehrt sind im Software Engineering gute Ergebnisse auch gelegentlich durch überragende Leistungen Einzelner zu erzielen, ohne dass jedoch eine hinreichende Systematik verfolgt wurde. Daher sollen sowohl Prozesse, als auch Ergebnisse in die Messung einfließen können.
- Da Software Engineering in einen komplexen organisatorischen bzw. betrieblichen Kontext eingebunden ist, sollen verschiedene Perspektiven auf einen Softwareentwicklungsprozess auch in die Analyse der Lernprozesse einbezogen werden. Daher soll SECAT sowohl Selbsteinschätzungen der Kompetenzentwicklung von Studierenden, als auch externe Blickwinkel wie die von Lehrenden oder Kunden ermöglichen. SECAT bietet dann eine Bewertungshilfe für Lehrende, um individuelle Kompetenzen bei Studierenden zu beurteilen, indem Einschätzungen aus unterschiedlichen Perspektiven miteinander verknüpft werden.
- Da Software Engineering üblicherweise im Team stattfindet, sind Kompetenzen für die Zusammenarbeit mit anderen zentral. Allerdings kann die Benotung von Studienleistungen an Hochschulen nur individuell erfolgen, auch wenn diese

Leistungen innerhalb eines Projektteams erbracht werden. Das entwickelte Messinstrument soll also sowohl individuelle, als auch Teamleistungen berücksichtigen.

- SECAT soll Kompetenzmessung zu einem bestimmten Zeitpunkt ebenso wie Deltaanalysen von Kompetenzen erlauben, die sich zwischen zwei Zeitpunkten entwickelt haben. So werden Kompetenzentwicklungsverläufe deutlich und vertiefen das Verständnis für Lehr-Lern-Prozesse und ihre Zusammenhänge (vgl. Sedelmaier & Landes, 2012).

4.2.4 Entwicklung des Kompetenzmodells

Ein vielversprechender empirisch und theoretisch fundierter Ansatz zur Kompetenzmessung ist in der beruflichen Bildung zu finden. Das dort zugrunde liegende Kompetenzmodell besteht aus den drei Kompetenzniveaustufen funktionale, prozessuale/konzeptuelle Kompetenz sowie ganzheitliche Gestaltungskompetenz, die durch acht Kriterien charakterisiert sind (vgl. Rauner, 2011). Dieses Kompetenzmodell wurde an die Besonderheiten des Software Engineering angepasst und erweitert (vgl. Abb. 2) (vgl. Sedelmaier & Landes, 2014b / Anhang 1.2).

Ergebnisse im Software Engineering sind meist Teamleistungen. Zudem sind Problembewusstsein und Lösungskompetenz im Sinne des Abwägens von Lösungsalternativen zentrale Kompetenzen (vgl. Abschnitt 4.1.4). Daher liegt der Fokus auf Zusammenarbeit und weiteren überfachlichen Kompetenzen des Kompetenzprofils SWEBOS, die sich nur schwer von außen beobachten lassen. Folglich setzt das Instrument SECAT – anders als der Ansatz von Rauner – vorwiegend auf Selbstbewertungen und Selbsteinschätzung. Dass diese Form der Evaluation ebenso valide und reliabel sein kann, zeigt BEvaKomp (vgl. Braun, 2008).

SECAT soll während des gesamten Studiums einsetzbar sein. In SECAT ist im Gegensatz zu Rauners Ansatz also eine sehr hohe Flexibilität erforderlich, da die Messzeitpunkte nicht wie bei Rauner immer Gesellenprüfungen sind, an denen es fest definierte Kompetenzen zu messen gilt. Und selbst innerhalb eines Studiengangs und auch bei verschiedenen Studiengängen, in denen Software Engineering gelehrt wird, variieren die zu messenden Kompetenzen stark.

Um mit SECAT studentische Kompetenzen im Software Engineering zu erfassen, wurde das ursprüngliche Kompetenzmodell nach Rauner (2011) auf Software Engineering angepasst, indem die acht Kriterien von der Berufsbildung im gewerblich-technischen Bereich auf Software Engineering übertragen wurden (vgl. Abb. 2). Mit

den acht angepassten Kriterien lässt sich das Kompetenzprofil SWEBOS komplett abbilden (vgl. auch Abschnitt 4.2.5).

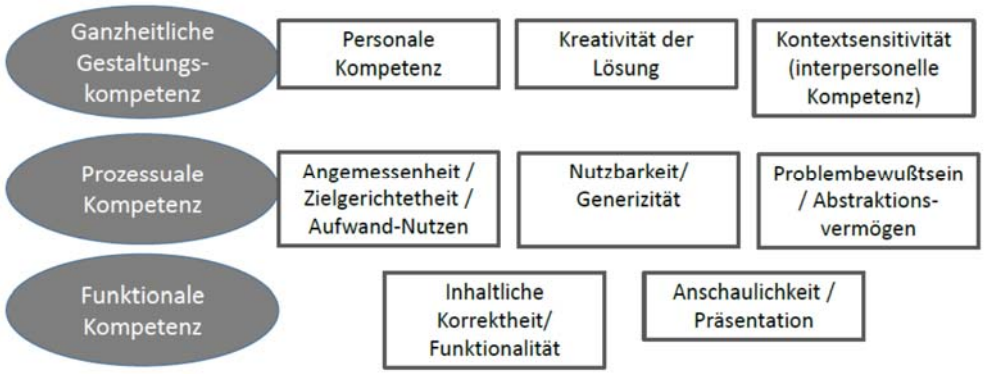


Abbildung 2 Kompetenzmodell in SECAT

Die Aufgliederung in die drei Niveaustufen funktionale, prozessuale und ganzheitliche Gestaltungskompetenz wurde beibehalten. Auch die beiden Kriterien für funktionale Kompetenz wurden nahezu unverändert übernommen, während die übrigen an die Besonderheiten des Software Engineering angepasst und erweitert wurden.

In Übereinstimmung mit Rauner lassen sich die Kriterien „Funktionalität“ und „Anschaulichkeit / Präsentation“ wie folgt beschreiben:

"(2) Funktionalität

Die Funktionalität einer vorgeschlagenen Lösung beruflicher Aufgaben ist ein auf der Hand liegendes Kernkriterium bei deren Bewertung. Die Funktionalität verweist auf die instrumentelle Fachkompetenz bzw. das kontextfreie, fachsystematische Wissen und die fachkundlichen Fertigkeiten. Der Nachweis der Funktionalität einer Lösungsvariante ist grundlegend und maßgebend für alle weiteren Anforderungen, die an Aufgabenlösungen gestellt werden." (Rauner, 2011, S. 57). Im Software Engineering handelt es sich hierbei etwa um das Beherrschen einer Programmiersprache.

"(1) Anschaulichkeit / Präsentation

Das Ergebnis beruflicher Aufgaben wird im Planungs- und Vorbereitungsprozess vorweg genommen und so dokumentiert und präsentiert, dass der/die Auftraggeber (Vorgesetzte, Kunden) die Lösungsvorschläge kommunizieren und bewerten können. Insofern handelt es sich bei der Veranschaulichung und Präsentation bzw. bei der Form einer Aufgabenlösung um eine Grundform beruflicher Arbeit und beruflichen Lernens. Eine zentrale Facette für die Kommunikation im Beruf ist die Fähigkeit, sich durch Beschreibungen, Zeichnungen und Skizzen und klar und strukturiert mitteilen zu können." (Rauner, 2011, S. 56)

Auf dem Niveau der prozessualen Kompetenz erfolgt die Beschreibung der Kriterien in Anlehnung an Rauner, jedoch mit Anpassung an Software Engineering:

Das Kriterium „Wirtschaftlichkeit“ wird mit Bezug auf Software Engineering konkretisiert. Während Rauner allgemein die "...kontextbezogene Berücksichtigung wirtschaftlicher Aspekte bei der Lösung beruflicher Aufgaben" (Rauner, 2011, S. 57) nennt, kommt im Software Engineering die Ausrichtung des zu entwickelnden Softwaresystems auf Kundenanforderungen stärker zum Tragen. Auch hier ist wie bei Rauner das Verhältnis von Aufwand und betrieblichem Nutzen zu berücksichtigen. Daher heißt dieses Kriterium in SECAT „Angemessenheit / Zielgerichtetheit / Aufwand-Nutzen-Verhältnis“.

Rauners Kriterium „Nachhaltigkeit / Gebrauchswertorientierung“ ist auf ein nicht real greifbares Produkt wie Software nicht direkt übertragbar und wird zum Kriterium „Nutzbarkeit / Generizität“ weiterentwickelt. Das Kriterium „Nutzbarkeit / Generizität“ im Software Engineering bezieht sich auf Aspekte, die die Qualität eines Softwaresystems beschreiben. Gerade auch im Software Engineering ist Qualität ein bedeutender Faktor und häufig nur schwer zu fassen (vgl. z.B. Reißing, 2015). Einen Beitrag dazu leistet im Software Engineering u.a. der Standard ISO/IEC 25010, der international anerkannt und weit verbreitet ist (vgl. ISO/IEC JTC 1/SC 25010:2011). Daher besitzt das Kriterium „Nutzbarkeit / Generizität“ im Software Engineering folgende Indikatoren:

- Vollständigkeit: Deckt die Lösung die wesentlichen Anforderungen des Auftraggebers / Kunden ab?
- Usability: Wie hoch ist die Bedienerfreundlichkeit der Lösung für Anwender?
- Robustheit: Wie unempfindlich ist die Lösung gegenüber Fehlbedienung oder anderen äußeren Störeinflüssen?
- Effizienz: Wie leistungsfähig ist die Lösung im Hinblick auf Laufzeit und Ressourcennutzung?
- Sicherheit: a) Ist das System vor unbefugtem Zugriff geschützt? (security) b) Vermeidet das System in ausreichendem Maß Schädigungen von Nutzern oder seiner Einsatzumgebung? (safety)
- Erweiterbarkeit: Wie einfach lässt sich die Lösung auf geänderte Anforderungen und Rahmenbedingungen anpassen und erweitern?
- Dokumentation, Strukturiertheit: Wie gut wird die Beseitigung / Korrektur von Fehlern und/oder die Wiederverwendung einzelner Softwarekomponenten in der Lösung unterstützt?

Auch Rauners Kriterium „Geschäfts- und Arbeitsprozessorientierung“ wird in SECAT modifiziert: Im Software Engineering genügt es nicht, lediglich vor- und nachgelagerte Prozesse und Schnittstellen bei der Lösung zu berücksichtigen. Vielmehr muss das Softwaresystem in ein komplexes Geflecht interagierender Faktoren eingepasst und mit einer Vielzahl an Rollen und Aufgaben innerhalb des Entwicklungsteams erstellt werden. Dabei ist es erforderlich, komplexe Vorgänge und Systeme sowie deren Zusammenhänge zu erkennen und zu verstehen. Da jedes Projekt anders ist, müssen auch fachliche Lösungsmöglichkeiten abgewogen und an die jeweilige Situation angepasst werden. Der Kontext, in dem die Aufgaben zu lösen sind, bedingt eine Vielzahl an technischen Herausforderungen für Software-Ingenieure, die in SWEBOS mit Problembewusstsein und Lösungskompetenz beschrieben werden. Dieses Kriterium wird zu „Problembewusstsein / Abstraktionsvermögen“.

Auch die Ebene der ganzheitlichen Gestaltungskompetenz erfordert Adaptionen. Rauners Kriterium „Sozialverträglichkeit“ ändert im Zusammenhang mit Software Engineering seine Bedeutung. Der Zusammenarbeit in und außerhalb eines Teams kommt eine stärkere Bedeutung zu als in der gewerblich-technischen Ausbildung. Software kann meist nicht alleine von einem einzelnen entwickelt werden. Vielmehr findet Softwareentwicklung in einem sehr komplexen Umfeld statt. Dies beinhaltet zum einen die Zusammenarbeit innerhalb eines Teams, in dem verschiedene Rollen ausgefüllt werden müssen und zielgerichtet und effektiv zusammengearbeitet werden muss. Zum anderen ist es auch erforderlich, außerhalb des Teams, etwa mit Kunden oder späteren Nutzern, zu kommunizieren. Es müssen immer auch die Rahmenbedingungen, der Kontext, berücksichtigt und mit diesem aktiv interagiert werden. Diese Kompetenz ist in SWEBOS als „Zusammenarbeit mit anderen Menschen“ (Z) näher erläutert und beinhaltet einen Blick über den Tellerrand der eigenen „Entwicklungsinsel“ hinaus. Hier unterscheidet sich das Kriterium vom Kriterium „Problembewusstsein / Abstraktionsvermögen“, das noch stark an der zu entwickelnden Software orientiert ist und auf komplexe technische Zusammenhänge abzielt, während das Kriterium „Kontextsensitivität / interpersonelle Kompetenz“ auch etwa interdisziplinäre Zusammenhänge berücksichtigt. Hier geht es nicht mehr um das Softwaresystem „im Kleinen“, sondern um den Kontext, in den dieses eingebettet ist. Im Kriterium „Problembewusstsein / Abstraktionsvermögen“ liegt der Fokus auf der zu entwickelnden Software und dem direkten Umfeld, während „interpersonelle Kompetenz“ größere Zusammenhänge berücksichtigt und auch beispielsweise Aspekte der Teamführung integriert. Daher wurde dieses Kriterium in SECAT zu „Kontextsensitivität / interpersonelle Kompetenz“.

Rauners Kriterium „Umweltverträglichkeit“ kommt im Zusammenhang mit physisch nicht greifbarer Software nur am Rande zum Tragen. Daher wurde es ersetzt durch ein

Kriterium, das auf die personalen überfachlichen Kompetenzen eines Software-Ingenieurs fokussiert, nämlich das Kriterium „intrapersonale Kompetenz“. Die Strukturierung in intra- und inter-personelle Kompetenzen findet sich ebenfalls im Kompetenzprofil wieder und hat sich dort als sinnvoll erwiesen. Inter-personelle Kompetenzen kommen im Umgang und bei der Zusammenarbeit mit anderen Menschen besonders zum Tragen, wie etwa kommunikative Kompetenzen. Dagegen liegen „intra-personale“ Kompetenzen in der Person und ihren Einstellungen und Werten selbst.

Das Kriterium „intrapersonale Kompetenz“ in SECAT beschreibt die in einer Person liegenden Fähigkeiten und Kompetenzen, die im Software Engineering erforderlich sind. Dazu gehören beispielsweise die Fähigkeit zur Strukturierung der eigenen Arbeit, Selbstreflexion oder auch Zeitmanagement (vgl. SWEBOS S und SWEBOS P) (vgl. Sedelmaier & Landes, 2015d / Anhang 1.6). Anders als interpersonelle Kompetenzen / Kontextsensitivität kommen intrapersonale Kompetenzen unabhängig von der Zusammenarbeit mit anderen zum Tragen.

Rauners Kriterium der Kreativität, in dem er auf die höchst unterschiedlichen Gestaltungsspielräume bei der Lösung beruflicher Aufgaben hinweist, wurde unverändert übernommen. Er weist darauf hin, dass "das Kriterium 'Kreative Lösung' in besonderer Weise berufsspezifisch interpretiert und operationalisiert werden" muss (Rauner, 2011, S. 58).

4.2.5 Operationalisierung von SECAT auf den konkreten Anwendungsfall

Das Kompetenzmodell im Messinstrument SECAT bildet das gesamte Kompetenzprofil im Software Engineering ab, das als Zielmarke der gesamten Hochschulausbildung im Software Engineering gilt. Üblicherweise werden allerdings in einzelnen Lehrveranstaltungen nur Teilbereiche des Kompetenzprofils adressiert und nicht alle globalen Kompetenzen, wie sie das umfassende Kompetenzprofil SWEBOS abbildet. Folglich ist es nötig, SECAT auf den konkreten Anwendungsfall etwa einer speziellen Lehrveranstaltung anzupassen. Es ist also ein Konkretisierungsschritt erforderlich, bevor eine Operationalisierung mit Items sinnvoll möglich ist.

Dieser Konkretisierungsschritt bringt die Kriterien des Kompetenzmodells und die jeweiligen Lehrziele einzelner Lehrveranstaltungen oder -einheiten in Deckung. Dabei werden die acht Kriterien spezifiziert. Dabei ist es nicht immer sinnvoll, dass alle Kriterien in einer Lehreinheit adressiert werden. Häufig findet eine Fokussierung statt, wodurch Kompetenzen Schritt für Schritt aufgebaut, in Lehreinheiten adressiert und individuelle Kompetenzentwicklungsverläufe nachverfolgt werden können. Wie bereits

erwähnt können nicht in einer einzigen Lehreinheit alle erforderlichen Kompetenzen adressiert werden. Zudem sind die Niveaustufen im Kompetenzmodell hierarchisch geordnet und bauen aufeinander auf. Ohne ein Verständnis für die grundlegenden fachkundlichen Fertigkeiten auf funktionaler Ebene kann etwa keine Kreativität der Lösung erreicht werden. Dies entspricht einem gängigen Kompetenzverständnis in der Pädagogik (vgl. z.B. Erpenbeck & Sauter, 2013, S. 32-33).

Beispielsweise adressiert eine Lehrveranstaltung im vierten Semester des Bachelorstudiengangs Informatik der Hochschule Coburg im Bereich der „Kontextsensitivität / interpersonelle Kompetenzen“ Kompetenzen, die erforderlich sind, um ein Kundengespräch zu führen. Dazu zählen z.B. Moderationstechniken, Gesprächsführung und Empathie, was sich auch im Kompetenzprofil wiederfindet. Im Gegensatz dazu konkretisiert sich Kontextsensitivität für einen Masterstudierenden, der ein Softwareentwicklungsprojekt leitet, auf andere Weise, nämlich eher durch Teamkompetenzen wie Motivationsfähigkeit oder Führen von Mitarbeitergesprächen. Auch diese Kompetenzen sind im Kompetenzprofil zu finden, und auch sie konkretisieren das Kriterium „Kontextsensitivität“, erhalten im speziellen Kontext jedoch eine eigene Ausprägung (vgl. Tab. 1).

Dabei ist nicht notwendigerweise jede Konkretisierung aus Tabelle 1 unmittelbar in SWEBOS zu finden, da SWEBOS dichte Beschreibungen enthält, die durch exemplarische Verhaltensweisen näher beschrieben werden und keinen Anspruch auf Vollständigkeit erheben. Die Handlungsanker in SWEBOS dienen dem Verständnis und sind nicht als Checkliste anzusehen. Software Engineering ist zu komplex, um jedes mögliche Verhalten in SWEBOS abzubilden.

Tabelle 1: **Kompetenzmodell und zugehörige Konkretisierungen im Software Engineering Competence Assessment Tool (SECAT)**

Kompetenzniveau *Kriterium*	***Konkretisierungen*** *(ggf. mit Verweis auf SWEBOS – vgl. Abschnitt 4.1.3)*
Ganzheitliche Gestaltungskompetenz	
Kontextsensitivität (interpersonelle Kompetenz)	• Moderation • Führen eines Kundengespräches • Platz im Team / in der Gruppe finden • Sensitivität / Offenheit für Empfindungen und Bedürfnisse anderer (Empathie) • Führen eines Entwicklungsteams • Sachliche Arbeitsweise / Verbindlichkeit
Personale Kompetenzen	• Durchhaltevermögen, am-Ball-bleiben, nachfragen, nachhaken (SWEBOS P) • Arbeitstechniken, Qualitätsbewusstsein, Sorgfalt (SWEBOS S) • Strukturiertes Vorgehen • Rollenverteilung in der Gruppe • Zeitmanagement (SWEBOS S), Termintreue • Kenntnis eigener Talente (SWEBOS Z) • Offenheit für Neues • Eigeninitiative, Motivation • Zielorientierung (SWEBOS S) • Selbstreflexion (SWEBOS P) • Frustrationstoleranz • Kompromiss- und Konfliktfähigkeit • Individuelle Teamfähigkeit • professionelle Zusammenarbeit mit anderen Menschen (SWEBOS Z)
Kreativität der Lösung	• Methodenvielfalt, um an Ergebnisse zu kommen • Berücksichtigung vorhandener Komponenten und Integration zu Lösung • Effizienz des Lösungswegs
Prozessuale Kompetenz	
Angemessenheit / Zielgerichtetheit / Aufwand-Nutzen-Verhältnis	• Anwendung methodischer Vorgehensweisen und Abstimmung auf den direkten Kontext • Aufwand-Nutzen-Abwägung • Zielgerichtetheit
Nutzbarkeit / Generizität	• Priorisierung von Anforderungen • Dokumentation • Weiterentwicklungsmöglichkeiten des Systems (Nachhaltigkeit)

	• Usability • Robustheit / Fehlerfreiheit des Systems • Passgenauigkeit des Systems
Problembewusstsein / Abstraktionsvermögen	• Abwägen von Lösungsalternativen • Verständnis komplexer Vorgänge, Systeme und Zusammenhänge (Problembewusstsein) (SWEBOS V) • Berücksichtigung nicht-funktionaler Anforderungen
Funktionale Kompetenz	
Inhaltliche Korrektheit / Funktionalität	Auf dieser Ebene existieren keine Konkretisierungen
Anschaulichkeit / Präsentation	

Selbst wenn die Konkretisierung dieselbe ist, kann sie trotzdem auf verschiedenen Niveaustufen liegen, abhängig von Vorwissen und Kontext. Moderation hat etwa für Studienanfänger eine andere und geringere Komplexität als für Masterstudierende und erfährt so eine andere Ausprägung.

Auch diese Konkretisierungen sind „nur“ ein Zwischenschritt und müssen auf den jeweiligen Anwendungsfall heruntergebrochen werden. Also: Wie äußert sich diese interpersonelle Kompetenz beim Führen eines Kundengesprächs? Hier findet eine Spezifizierung auf die Lehrziele statt (vgl. Abb. 3). Stehen die Konkretisierungen der jeweiligen Kriterien über Lehrziele dann fest, werden Items entwickelt, die diese Konkretisierungen weiter operationalisieren.

Abbildung 3 Vorgehensweise bei der Item-Entwicklung

So wird eine Bewertung aus unterschiedlichen Blickwinkeln mit verschiedenen Fragebögen für Studierende, Dozenten oder Projektkunden möglich. Aus Perspektive der Studierenden können sowohl Selbsteinschätzungen stattfinden, als auch Fremdbewertungen von Peers erhoben werden. Zudem ist es auf dieser Ebene möglich, neben Items, die die Teamleistung beurteilen, auch Items zu entwickeln, die individuelle Kompetenzentwicklung bewerten.

Die Anzahl der Kriterien bzw. Items je Konkretisierung bzw. Kriterium erlaubt es außerdem, die Kompetenzen z. B. für verschiedene Zielgruppen unterschiedlich zu gewichten. So wurden etwa die einzelnen Kriterien für die Bewertung der Projektleiter eines Software-Projektes wie in Tabelle 2 gezeigt gewichtet. Dabei wurde ihr Kompetenzzuwachs durch die Lehrenden, die Kunden des Projekts sowie seine Teammitglieder eingeschätzt. Da die Teammitglieder aufgrund mangelnder Berufserfahrung keine Aussagen zur Kreativität der Lösung bzw. zur Nutzbarkeit des Systems treffen konnten, wurden keine Fragen aus diesem Bereich gestellt.

Tabelle 2: **Gewichtung der Kriterien mittels Anzahl der Items am Beispiel eines Teamleiters**

Kriterium	**Anzahl der Fragen je Perspektive**			
	Lehrende	***Kunde***	***Team-mitglieder***	***Summe***
Funktionale Kompetenz				
Inhaltliche Korrektheit/Funktionalität	4	2	4	**10**
Anschaulichkeit/Präsentation	2	3	3	**8**
Prozessuale Kompetenz				
Angemessenheit/Zielgerichtetheit/Aufwand-Nutzen-Verhältnis	4	6	11	**21**
Nutzbarkeit/Generizität	4	5	0	**9**
Problembewußtsein/ Abstraktionsvermögen	4	2	5	**11**
Ganzheitliche Gestaltungskompetenz				
Kontextsensitivität (interpersonelle Kompetenz)	6	6	3	**15**
Personale Kompetenz	6	1	7	**14**
Kreativität der Lösung	4	2	0	**6**
Summe der Items	***34***	***27***	***33***	***94***

Die Items können in perspektivenspezifischen Fragebögen zusammengefasst und abgefragt werden, wofür eine Likert-Skala geeignet erscheint. Für alle hier vorliegenden Beispiele und Ergebnisse wurde eine 4-stufige Skala verwendet.

Die Anzahl der Fragen je Kriterium bzw. Konkretisierung ermöglicht eine Gewichtung der Kriterien innerhalb der Evaluation. Diese quantitative Evaluation ermöglicht eine statistische Weiterverarbeitung und Vergleichbarkeit der Ergebnisse.

Die Darstellung der Ergebnisse erfolgt üblicherweise auf Ebene der Kriterien oder innerhalb der gleichen Fragewelle auf Ebene der Konkretisierungen. Auf Ebene der Kriterien laufen alle Perspektiven der Bewertung zusammen. Die Kriterien bilden den festen, strukturgebenden Rahmen. Anpassung und Variabilität findet auf Ebene der Items und ggf. auf der Stufe der Konkretisierungsebene statt, auf der wiederum die Items aggregiert sind.

Der Vorteil der beschriebenen und in SECAT angewandten Methodik liegt in der Vergleichbarkeit über verschiedene Blickwinkel und Zeitpunkte hinweg bei ausreichender Anpassungsmöglichkeit an den konkreten Anwendungsfall.

4.2.6 Ergebnisse der Kompetenzbewertung mit SECAT

4.2.6.1 Ergebnisse einzelner Studierender

Beispielhaft werden hier Ergebnisse der Evaluation aus einem Software-Engineering-Projekt der Hochschule Coburg angeführt. In diesem Lehrformat arbeiten Bachelor-Studierende in ihrem letzten Studiensemester in Teams zusammen, die von jeweils einem Masterstudierenden geleitet werden (vgl. Sedelmaier & Landes, 2014c). Die Lehrziele sind für die beiden Studierendentypen unterschiedlich. Somit muss auch die Kompetenzmessung angepasst werden, etwa auf die Lehrziele des Masterstudierenden.

Das Evaluationsinstrument SECAT kann Kompetenzen einzelner Studierender und deren Entwicklung messen. Wie erwähnt sind neben der Selbstbewertung durch die Studierenden selbst weitere Perspektiven möglich. So ist es im Bildungsbereich erforderlich, dass Lehrende die Kompetenzen ihrer Studierenden einschätzen und benoten. Des Weiteren können mit SECAT Teammitglieder, Teamleiter oder auch Kunden in die Kompetenzmessung einbezogen werden. Als Beispiel sei hier das Kompetenzprofil eines Masterstudierenden eines Softwareentwicklungsprojektes aufgeführt (Abb. 4). Es zeigt die Perspektive eines Lehrenden, eines Kunden sowie die Perspektive des Teams, das der Masterstudierende geleitet hat. Der Masterstudierende erhielt ein 360°-Feedback von seinen Teammitgliedern, also fünf Bachelorstudierenden. So ist es mit SECAT möglich, dem Masterstudierenden ein umfassendes Feedback zu geben, das deutlich über die Sicht eines einzelnen Lehrenden hinausgeht. Dieses Vorgehen liefert auch einen Beleg für die Validität des Messinstruments, da die erhobenen Daten auf eine hohe Interrater-Reliabilität hinweisen, die sich in Abbildung 4 in der räumlichen Nähe der Kurven der einzelnen Perspektiven zeigt.

Die individuellen Kompetenzen können auf verschiedene Art und Weise dargestellt werden, etwa wie in Abbildung 4 als Kiviat-Diagramm. Ebenso denkbar sind jedoch Darstellungen mittels Boxplots oder Liniendiagrammen.

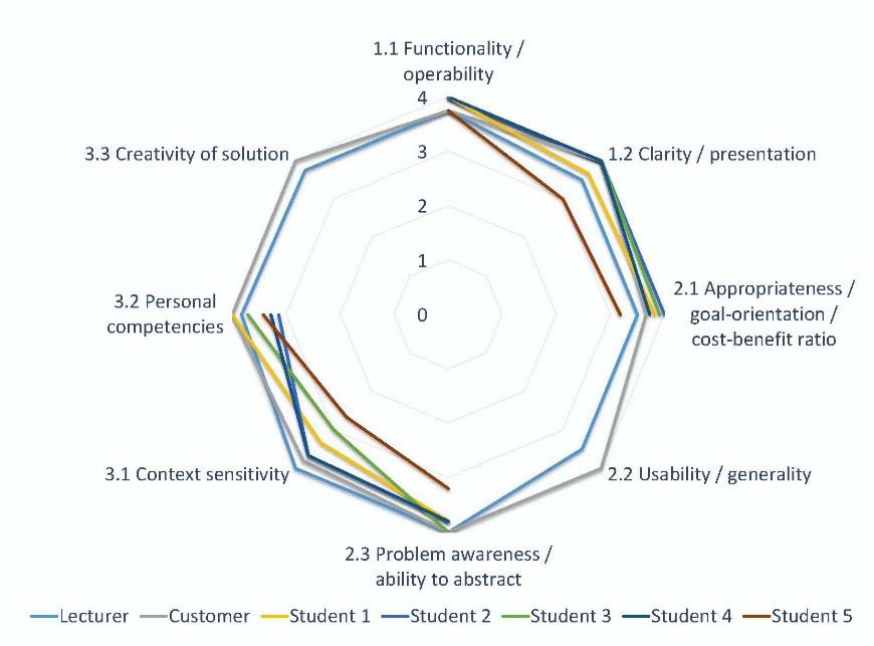


Abbildung 4 Kompetenzprofil eines Masterstudierenden bewertet aus verschiedenen Blickwinkeln

Für SECAT liegen zudem erste Hinweise auf eine hohe interne Konsistenz vor (Abb. 5).

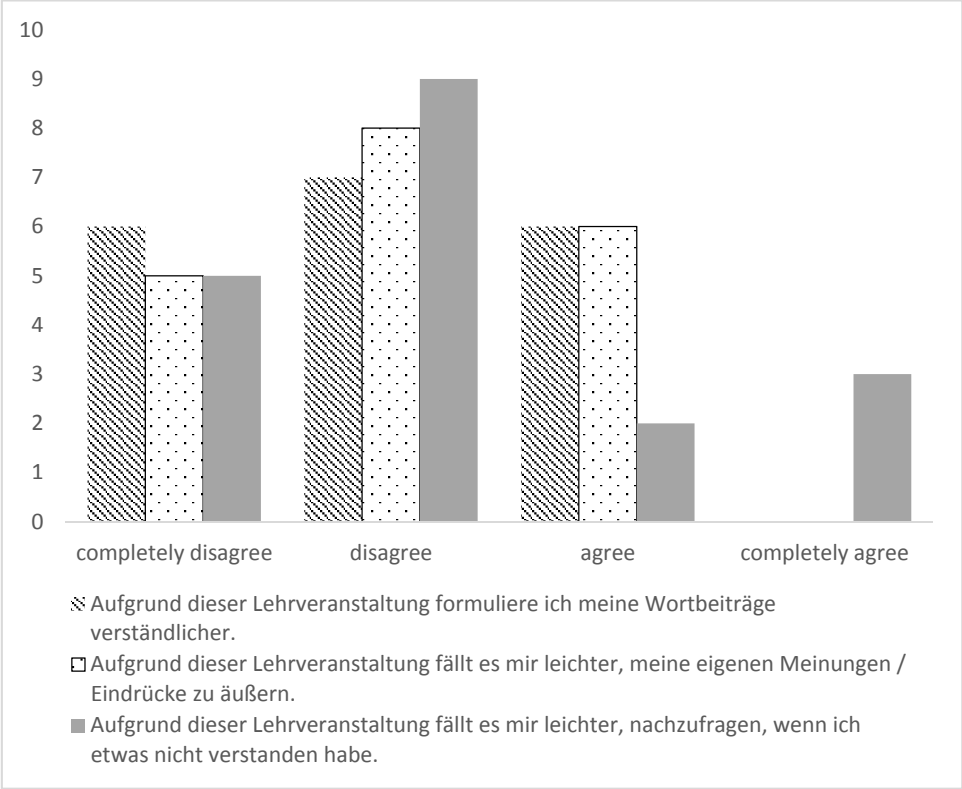


Abbildung 5 Interne Konsistenz von SECAT

4.2.6.2 *Ergebnisse im Hinblick auf die Weiterentwicklung kompetenzorientierter Lehre im Software Engineering*

Werden die Ergebnisse der Kompetenzmessung mehrerer Studierender, die dieselbe Lehrveranstaltung besucht haben, zusammengenommen, können etwa über Mittelwerte oder Mediane Rückschlüsse auf die Wirksamkeit der Lehre und der Lehrmethoden gezogen werden. Wenn mehrere Fragebögen zusammengenommen werden, mildern sich individuelle Ausreißer und extreme Sichtweisen ab. Auch hier zeigen Ergebnisse bereits deutlich, dass diejenigen Studierenden, die an den evaluierten Lehrmethoden teilgenommen haben, einen Kompetenzzuwachs in den adressierten Kompetenzen sehen (Abb. 6).

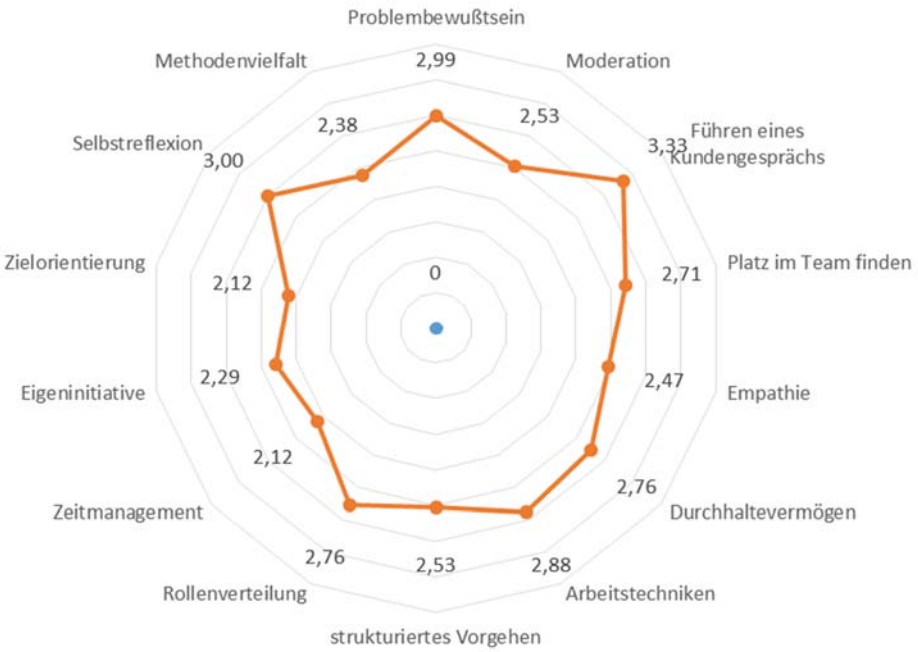


Abbildung 6 Mittlerer Kompetenzzuwachs aufgrund der Lehreinheit je Konkretisierung

Abbildung 6 zeigt Daten aus einer Lehrveranstaltung im dritten Fachsemester des Studiengangs Informatik der Hochschule Coburg. Hier sollten die Studierenden in einem Kundengespräch Anforderungen an eine zu entwickelnde Software erheben (vgl. Sedelmaier & Landes, 2014a / Anhang 1.1). Dazu wurden sie mit einem konkreten Arbeitsauftrag in die Vorbereitung eines solchen Kundengesprächs und dann in das Gespräch selbst geschickt. Die adressierten Lehrziele stammten in diesem Fall vornehmlich aus den Kriterien Problembewusstsein, Kontextsensitivität, Personale Kompetenz und Kreativität der Lösung. Diese Kriterien wurden in den folgenden Bereichen (Tab. 3) konkretisiert, die sich ebenso in der Auswertung (Abb. 6) wiederfinden:

Tabelle 3: **Anpassung an konkrete Lehrziele am Beispiel**

Kriterium	**Konkretisierung**
Prozessuale Kompetenz	
Problembewusstsein / Abstraktionsvermögen	• Fähigkeit, komplexe Vorgänge und Systeme sowie Zusammenhänge zu verstehen (Problembewusstsein) (SWEBOS V)
Ganzheitliche Gestaltungskompetenz	
Kontextsensitivität (interpersonelle Kompetenz)	• Moderation • Führen eines Kundengesprächs • Platz im Team finden • Sensitivität / Offenheit für Empfindungen und Bedürfnisse anderer (Empathie)
Personale Kompetenz	• Arbeitstechniken (SWEBOS S) • Strukturiertes Vorgehen • Rollenverteilung in der Gruppe • Zeitmanagement (SWEBOS S) / Termintreue • Eigeninitiative / Motivation • Zielorientierung (SWEBOS S) • Selbstreflexion (SWEBOS P)
Kreativität der Lösung	• Methodenvielfalt, um an Ergebnisse zu kommen

Die stärkste Gewichtung lag mit 10 Fragen auf der Kontextsensitivität, gefolgt von personalen Kompetenzen (8 Fragen), Problembewusstsein mit 4 Fragen und Kreativität der Lösung mit 2 Fragen.

In diesem konkreten Fall wurde nur die individuelle Selbsteinschätzung der Studierenden als Perspektive verwendet.

Bei genauerer Analyse der Daten lässt sich feststellen, dass besonders diejenigen Studierenden (Nr. 3, 5 und 10), die nicht an allen entsprechenden Lehreinheiten teilgenommen hatten, einen geringeren Kompetenzzuwachs in dort spezifisch adressierten Bereichen angeben als die meisten ihrer KommilitonInnen (Abb. 7).

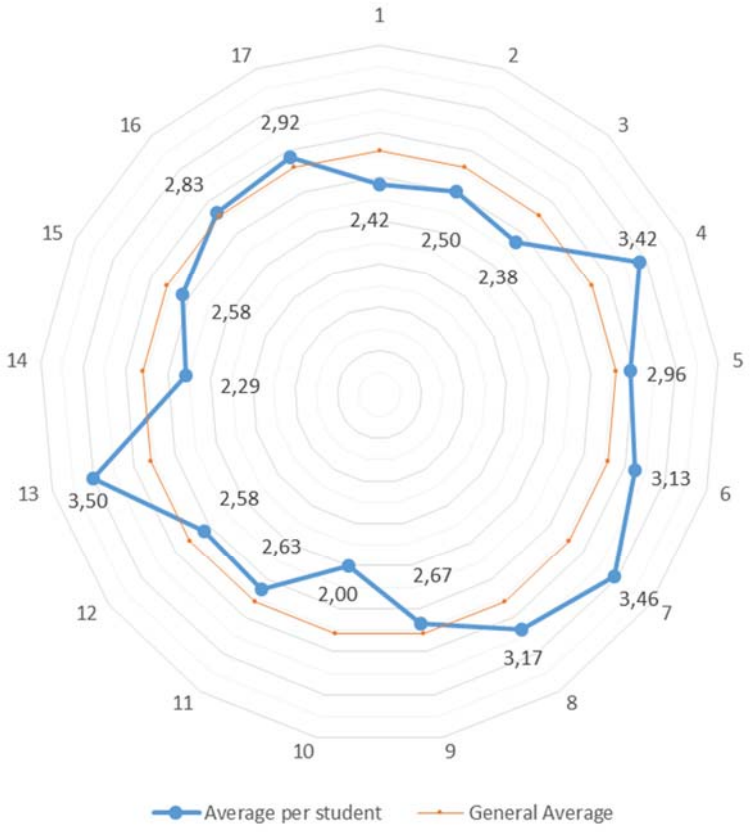


Abbildung 7 Kompetenzzuwachs in den adressierten Kriterien je Studierender

Insgesamt jedoch weisen die Studierenden in den meisten Fällen einen deutlichen Kompetenzzuwachs auf (Abb. 8).

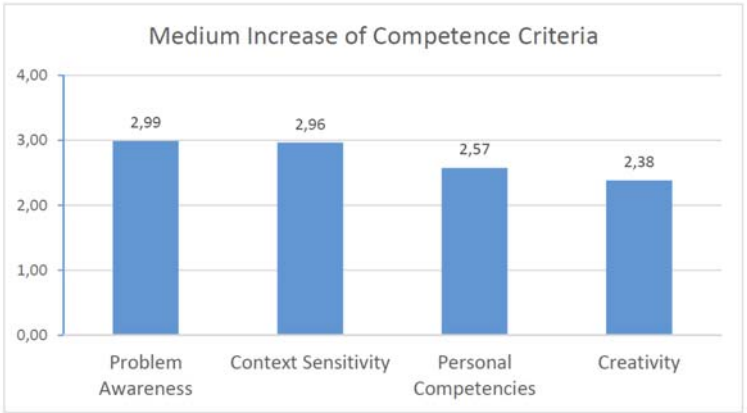


Abbildung 8 Mittlerer Kompetenzzuwachs je Kriterium

Die vorliegenden Ergebnisse zeigen deutlich, dass sich besonders aus kumulierten Werten mehrerer Studierender derselben Lehreinheit Rückschlüsse auf die Wirksamkeit von Lehrkonzepten ziehen lassen. Ein besseres Verständnis, welche

didaktischen Konzepte gut funktioniert haben und welche noch Verbesserungspotential aufweisen, erlaubt auch Rückschlüsse auf abstrakterer Ebene. Daraus resultiert ein besseres Verständnis für Lehr-Lernprozesse im Software Engineering, das die Grundlage für eine Fachdidaktik für Software Engineering bildet.

4.2.7 Zusammenfassung und Ausblick

SECAT ist ein Messinstrument zur kompetenzorientierten Weiterentwicklung der Lehre von Software Engineering, das vom Kompetenzzuwachs der Studierenden Rückschlüsse auf die Wirksamkeit der Lehrkonzepte erlaubt.

Ausgehend vom Software-Engineering-spezifischen Kompetenzprofil SWEBOS und einem Kompetenzmessungsansatz in der beruflichen Bildung (vgl. Rauner, 2011) wurde für die Kompetenzmessung Studierender und die darauf aufbauende Evaluation von Lehrveranstaltungen ein eigener Evaluationsansatz sowie ein eigenes Kompetenzmodell entwickelt. Letzteres bildet den Rahmen für das zu erreichende Kompetenzprofil aus fachlichen Kenntnissen und insbesondere kontextsensitiven überfachlichen Kompetenzen. Das Evaluationsinstrument der Fachdidaktik des Software Engineering baut auf diesem Kompetenzmodell auf und berücksichtigt auch die Besonderheiten des Fachs, um die Wirksamkeit von didaktischen Konzepten in diesem speziellen Fach zu evaluieren. Gradmesser sind die Kompetenzen, die Studierende aufgrund einer Lehrveranstaltung gewonnen haben. Diesen Zuwachs gilt es zu erheben, um daraus Rückschlüsse auf die Lehrqualität zu ziehen.

In zukünftigen Studien gilt es, das Instrument SECAT weiter zu validieren. Ein wesentliches Ziel besteht darin zu validieren, dass sich das Kompetenzprofil vollumfänglich im Messinstrument widerspiegelt. Ein weiterer wesentlicher Aspekt ist die tiefergehende Prüfung des Messinstruments auf Reliabilität und Validität, etwa durch qualitative Forschungsmethoden. Diese Validierung bezieht sich zum einen auf die Prüfung der einzelnen Items. Zum anderen soll sichergestellt werden, dass auch das Messinstrument als Ganzes valide für die individuelle Kompetenzmessung bei Studierenden und für die Bewertung der Qualität der Lehre ist. Ein wesentlicher Aspekt hierbei ist auch die Sicherstellung der internen Konsistenz des Messinstruments, also die Definition von Qualitätskriterien für die Widerspruchsfreiheit von Items einerseits und Rückläufen bei der Kompetenzmessung andererseits. Um die interne Konsistenz beurteilen und das Messinstrument ggf. weiter verbessern zu können, sind Kriterien zu entwickeln, die einen Rückschluss auf mögliche Widersprüchlichkeiten zulassen. Daneben sind weitere Kriterien erforderlich, die die interne Konsistenz der Antworten von Befragten messen, um ggf. bei der Auswertung offenkundig wenig sinnvolle Befragungsergebnisse erkennen und entsprechend verarbeiten zu können. Ein

möglicher Ansatzpunkt ist hier die Entwicklung von Maßzahlen ähnlich dem Consistency Index oder der Consistency Ratio innerhalb des multi-kriteriellen Entscheidungsverfahrens AHP (Analytic Hierarchy Process) (vgl. Saaty, 2001).

Zur Validierung von SECAT ist eine Ausweitung der Datenbasis erforderlich. Auch sind qualitative Validierungen angedacht. Dabei könnten die Ergebnisse aus SECAT etwa mit handlungsorientierten (z. B. Kasseler-Kompetenz-Raster (Kauffeld, 2002)) oder biographieorientierten Verfahren (z. B. ProfilPass (Bretschneider et al., 2007; Deutsches Institut für Erwachsenenbildung, 2012)) untermauert werden. In einem nächsten Schritt kann dann die Übertragbarkeit von SECAT auf Hochschulfächer mit ähnlichen Herausforderungen geprüft werden. Dazu gehört etwa die Didaktik für höhere Mathematik im Hochschulbereich.

Eine Ausweitung der Datenbasis erfordert geeignete Werkzeugunterstützung. In diesem Zusammenhang ist es sinnvoll, die bereits als Prototyp entstandene Evaluationssoftware weiterzuentwickeln, die das Erstellen von Befragungen vereinfacht und verschiedene Auswertungsmöglichkeiten bietet. Zudem erleichtert diese dann die Auswertung und Vergleichbarkeit der Daten.

4.3 (Weiter-)Entwicklung von Lehr-Lern-Konzepten im Software Engineering

4.3.1 Organisatorisches Umfeld

Der Hauptteil der Lehrveranstaltungen, auf denen die beschriebenen Forschungen basieren, fand an der Hochschule für Angewandte Wissenschaften Coburg statt. Da es sich bei dem Forschungsprojekt um ein Verbundprojekt handelt, wurde ebenso eine Lehrveranstaltung im Studiengang Mechatronik der Hochschule Aschaffenburg pädagogisch betreut.

Die meisten der in dieser Arbeit betrachteten Lehrveranstaltungen sind an der Hochschule Coburg im Bachelor-Studiengang Informatik angesiedelt. Ab dem dritten Semester können die Studierenden Wahlpflichtfächer aus drei Vertiefungsrichtungen wählen. Neben Embedded Systems und Wirtschaftsinformatik wird auch die Richtung Softwaretechniken angeboten, deren Kern Software Engineering darstellt.

Zu Beginn des Dissertationsvorhabens standen lediglich die Lehrveranstaltungen aus dem Software Engineering im Fokus. Dazu gehört eine „klassische“ Vorlesung Software Engineering im dritten Semester, die einen Überblick über das Themenfeld Software Engineering gibt und für alle Studierenden eine Pflichtveranstaltung mit vier Semesterwochenstunden (SWS) im Wintersemester ist. Diese Lehrveranstaltung beinhaltet ein Minipraktikum von ca. vier Wochen Dauer und je vier Stunden pro Woche. Im Schnitt wird Software Engineering von 40 bis 60 Studierenden besucht.

Im darauf folgenden Sommersemester wird ein Wahlpflichtfach Software-Modellierung und -Architektur angeboten. Dieses wird von zwei Lehrenden abgehalten, wobei im Rahmen der vorliegenden Arbeit der Fokus auf dem Teilbereich Software-Modellierung liegt, der zwei SWS umfasst. Im Schnitt nehmen hier ca. 20 bis 25 Studierende aus dem vierten bzw. sechsten Semester teil.

Bachelorstudierende haben dann in ihrem siebten Studiensemester die Möglichkeit, am Wahlpflichtfach Software Engineering Projekt teilzunehmen. Inhaltliche Voraussetzung hierfür ist allerdings der erfolgreiche Besuch von Software-Modellierung und -Architektur. In diesem Projekt sollen die Studierenden das Wissen und die Kompetenzen, die sie sich in ihrem bisherigen Studium stückweise angeeignet haben, zueinander in Bezug bringen und praktisch anwenden. Sie sollen so den gesamten Software-Entwicklungs-Zyklus am Stück in einem Projekt durchlaufen und so erfahren, wie die einzelnen Wissenselemente ineinandergreifen und zusammenhängen. Diese Lehrveranstaltung ist mit 180 Soll-Stunden relativ zeitaufwändig; viele Studierende investieren aber freiwillig noch mehr Zeit. Meist formieren sich je Durchgang zwei bis drei Projektteams mit je fünf bis sieben Bachelor-Studierenden.

Angeleitet werden die Bachelor-Studierenden im Software-Engineering-Projekt von einem Masterstudierenden aus dem Studiengang Informationstechnologie für Unternehmensanwendungen. Die Lehrziele sind im Gegensatz zu denjenigen der Bachelor-Studierenden stark auf überfachliche Kompetenzen wie Führen eines Entwicklungsteams oder Kommunikation innerhalb und außerhalb des Teams ausgerichtet. Dieses Projekt ist für die Masterstudierenden ein Wahlpflichtfach mit ebenfalls 180 Soll-Stunden.

Später wurde die Einführungsveranstaltung „Grundlagen der Informatik“ im ersten Semester des Bachelorstudiengangs ebenfalls einer didaktischen Weiterentwicklung unterzogen, um die Basis für die späteren Software-Engineering-Veranstaltungen zu legen. Bei „Grundlagen der Informatik“ handelt es sich um eine umfangreiche Pflichtveranstaltung mit sechs SWS, in der theoretische und abstrakte Grundlagen des Fachs Informatik vermittelt werden.

Während in den ersten Semestern des Bachelorstudiengangs Informatik eher funktionales, „handwerkliches“ Wissen vermittelt werden soll, nehmen im Studienverlauf die überfachlichen Kompetenzen mit besonderer Berücksichtigung der kontextsensitiven überfachlichen einen immer stärkeren Raum ein.

Ferner resultieren einzelne Erkenntnisse in dieser Arbeit aus einer Lehrveranstaltung „Information and Communication Systems“ des internationalen Masterstudiengangs „Financial Management“ der Fakultät Wirtschaft an der Hochschule Coburg. Die

Studierenden kommen hier aus allen Teilen der Welt, häufig aus dem asiatischen Raum, und bringen unterschiedlichste Vorerfahrungen mit. In einer zweistündigen Lehrveranstaltung, die für alle Studierenden des Studiengangs Pflicht ist, sollen sie ein grundlegendes Verständnis für Informations- und Kommunikationssysteme erhalten.

4.3.2 Didaktisches Vorgehen bei der Entwicklung von Lehr-Lern-Konzepten

Diese Lehrveranstaltungen sollen die für Software Engineering spezifischen fachlichen und überfachlichen Kompetenzen fördern, die in SWEBOS beschrieben werden (vgl. Abschnitt 4.1.3). SWEBOS fungiert dabei als Kompass für die Hochschulausbildung. Um diese Ziele anzusteuern, werden die im vorherigen Abschnitt beschriebenen vorhandenen Lehrveranstaltungen didaktisch analysiert und kompetenzorientiert weiterentwickelt. Dabei finden allgemeindidaktische Theorien in den Lehrkonzepten Anwendung. Das Vorgehen bei der didaktischen Planung von Lehrveranstaltungen ist also zielorientiert (vgl. Mager, 1992, S. ix) und am Kompetenzprofil SWEBOS ausgerichtet.

Ausgangspunkt für eine spezifische Lehrveranstaltung sind immer die Lehrziele. Diese sind stark durch die Berufserfahrung des Lehrenden geprägt, orientieren sich an fachlichen Richtlinien wie etwa dem „Software Engineering Body of Knowledge“ (SWEBOK) (Bourque & Fairley, 2014) und leiten sich aus SWEBOS ab. Erst dann erfolgt eine Analyse des vorhandenen didaktischen Konzepts und didaktische Konzeptionierung. Das Vorgehen ist in Sedelmaier und Landes (2015c) (Anhang 1.5) detailliert beschrieben und soll daher an dieser Stelle zusammengefasst sowie mit einigen Beispielen untermauert werden.

Lehr-Lern-Forschung und allgemeine Didaktik stehen häufig in einem konträren Verhältnis (vgl. Terhart, 2002). Der Lehr-Lern-Forschung wird vorgeworfen, sie ignoriere allgemeindidaktische Theorien (vgl. Achtenhagen, 2006, S. 591). Im vorliegenden interdisziplinären Forschungsansatz soll dieser Gegensatz aufgebrochen und eine Verbindung zwischen allgemeindidaktischen Theorien und einer Fachdidaktik für Software Engineering hergestellt werden. Dazu wurden allgemeintheoretische didaktische Modelle analysiert und auf ihre Übertragbarkeit auf eine Fachdidaktik für Software Engineering überprüft, wie in Sedelmaier und Landes (2015c) (Anhang 1.5) beschrieben. Elemente, die sinnvoll und zielführend scheinen, werden übernommen und mit Aspekten anderer allgemeindidaktischer Theorien verknüpft sowie um neue Aspekte auch aus der Lehr-Lern-Forschung ergänzt, um daraus ein stimmiges Vorgehen bei der Planung und Organisation sowie Evaluation von Lehrveranstaltungen im Software Engineering zu erhalten (Abb. 9).

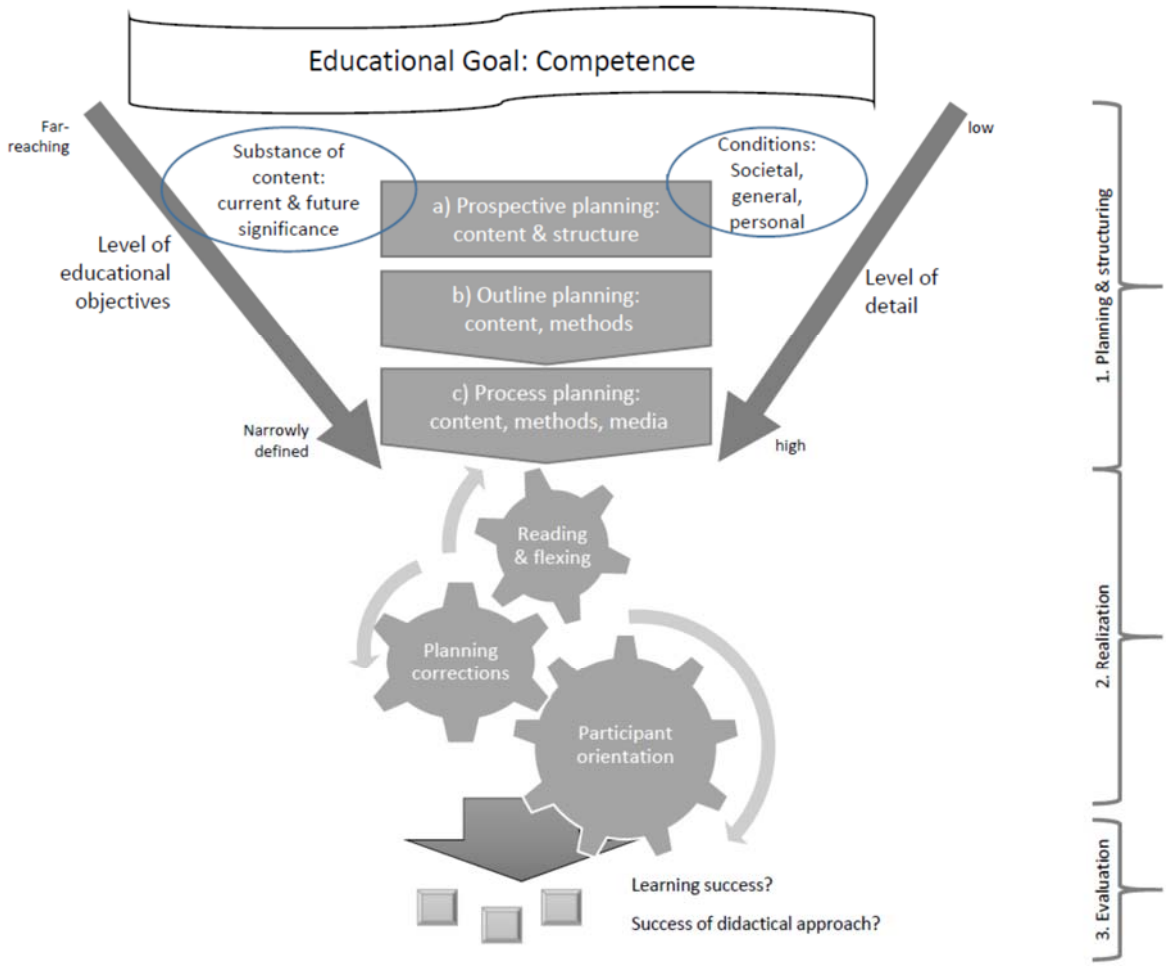


Abbildung 9 Systematisches Vorgehen zur Planung, Konzeption, Umsetzung und Auswertung von Lehrkonzepten nach Sedelmaier und Landes (2015c)

Aus Klafkis Didaktischer Analyse (vgl. Klafki, 1964) fließen bildungstheoretische Aspekte ein, die die systematische Analyse von Zielen und Inhalten hinsichtlich ihres Bildungsgehalts bei der Lehrveranstaltungsplanung erleichtern. Ferner folgt der Ansatz in diesem Vorhaben dem Primat der Didaktik vor methodischen Entscheidungen.

Aus der Lehrtheoretischen Didaktik wird die Berücksichtigung der Ausgangslage und der Rahmenbedingungen sowie die Entscheidungsfaktoren (vgl. Heimann, Otto & Schulz, 1965) übernommen und angewandt.

Der vorliegende Ansatz von Sedelmaier und Landes (2015c) (Anhang 1.5) basiert wie in Abschnitt 3.3 beschrieben auf konstruktivistische Grundannahmen. Daraus kommen Aspekte der Teilnehmerorientierung (vgl. Siebert, 1997, S. 105) ebenso wie aktivierende Methoden hinzu.

Neu ist – anders als bei vorliegenden allgemeindidaktischen Theorien – neben der ausgeprägten Zielorientierung die stärkere Betonung der Evaluation von Lehr-Lern-Konzepten zur Wirksamkeitsprüfung, was als Brückenschlag zur Lehr-Lern-Forschung

gesehen werden kann. Das beschriebene Vorgehen bei der Lehrveranstaltungsplanung vereint somit die top-down-Theorien der allgemeinen Didaktik mit den bottom-up-Forschungsergebnissen, die eher in der Lehr-Lern-Forschung zu finden sind.

4.3.3 Beispielhafte Umsetzung einer zielorientierten Lehrkonzeptentwicklung

Dieses Vorgehen zur Analyse und Planung sowie Evaluation von Lehrveranstaltungen wurde vor allem in den in Abschnitt 4.3.1 beschriebenen Lehrveranstaltungen angewandt. Erfahrungen damit wurden an anderen Stellen publiziert, so dass hier wiederum eine Zusammenfassung der wesentlichen Elemente gegeben wird. Alle im Folgenden beschriebenen Lehrkonzepte starten mit einer expliziten Analyse der Lehrziele und einer Anbindung an SWEBOS. Daher wird im weiteren Verlauf verzichtet, darauf hinzuweisen.

4.3.3.1 Software Engineering

Eine Publikation zeigt die Kombination einzelner Methoden zu einem zielorientierten und schlüssigen Gesamtkonzept (vgl. Sedelmaier & Landes, 2015b / Anhang 1.4). Die Studierenden werden Schritt für Schritt an das Thema Software Engineering und die dort zu bewältigenden Problemstellungen herangeführt. Durch induktive und aktivierende Lehrmethoden erarbeiten sie sich Lösungsmöglichkeiten aktiv. Zusammen mit den fachlichen Inhalten werden überfachliche Kompetenzen adressiert und aufgebaut. Ohne überfordert zu werden, übernehmen die Studierenden zunehmend Verantwortung für ihren eigenen Lernprozess.

In dieser Lehrveranstaltung liegt der Schwerpunkt der adressierten Kompetenzen auf funktionaler und prozessualer Ebene, da es sich um eine verpflichtende Einführung in Software Engineering handelt. Auf prozessualer Ebene ist das Kriterium „Problembewusstsein“ angesiedelt, was als zentrales Lehrziel definiert wurde. Die Ebene der ganzheitlichen Gestaltungskompetenz kommt erst in den späteren Semestern vermehrt zum Tragen. Die Evaluation des didaktischen Ansatzes mit SECAT zeigt deutliche Kompetenzzuwächse in den adressierten Kompetenzen auf funktionaler und prozessualer Ebene (Abb. 10).

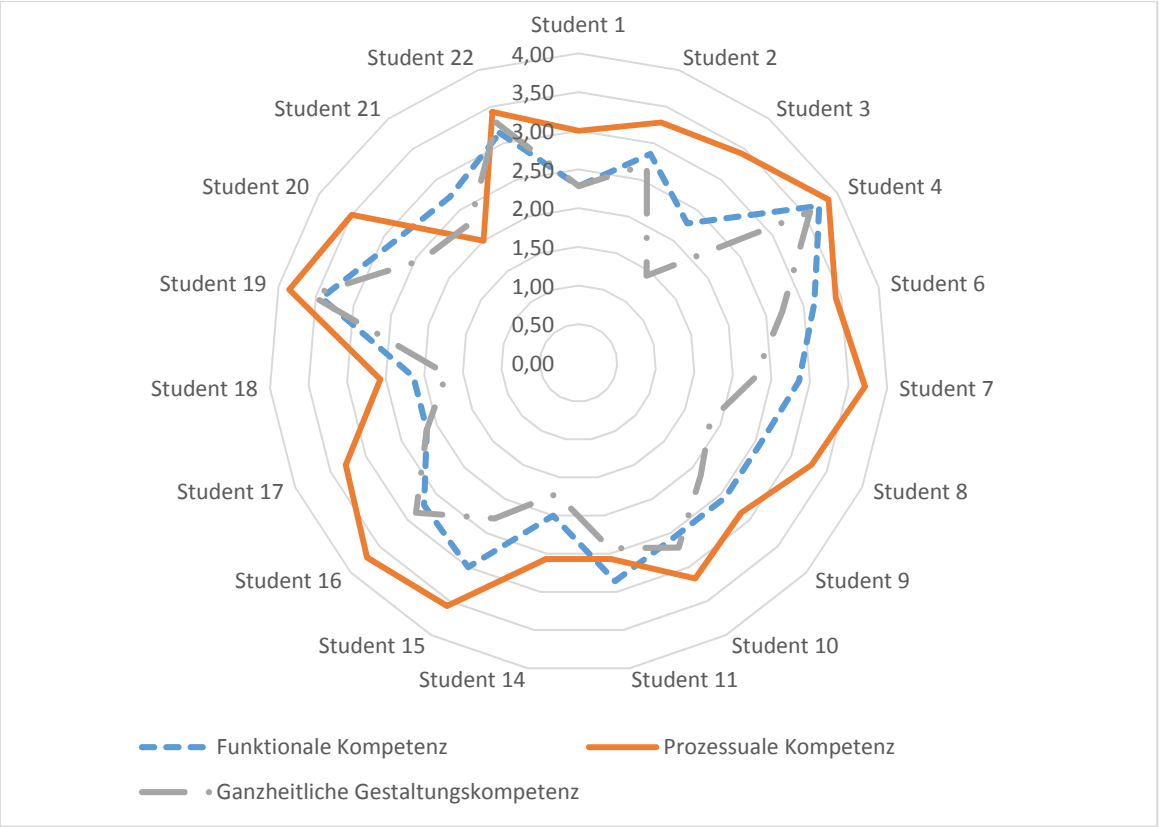


Abbildung 10 Kompetenzzuwachs je Studierender und Kompetenzniveau

4.3.3.2 *Software Modellierung und -Architektur, Teil Modellierung, erste Überarbeitungsphase*

Die Analyse und Weiterentwicklung des Lehrkonzepts in dieser Phase wird in Sedelmaier und Landes (2014e) (Anhang 1.3) im Detail erläutert. In dieser Lehrveranstaltung trat besonders das Problem zutage, dass die Studierenden keine Herausforderung darin sahen, Anforderungen an ein zu entwickelndes Softwaresystem zu erheben. Diese Anforderungen liegen normalerweise nicht explizit vor und sind nur schwerer als zunächst angenommen zu erheben. Als eine mögliche Ursache kann angeführt werden, dass die Kunden eben in der Regel keine Informatiker sind und die Studierenden somit lernen müssen, über Disziplingrenzen hinweg mit Kunden zu kommunizieren. Die Studierenden unterschätzen häufig die Schwierigkeiten, sich in ein fachfremdes Gegenüber mit seiner spezifischen Fachterminologie hineinzuversetzen. Dabei haben Kunden häufig eine eigene, andere Sicht auf das zu entwickelnde Softwaresystem und setzten völlig andere Prioritäten als Software-Ingenieure. Diese Schwierigkeit bei der interdisziplinären Zusammenarbeit beschreibt auch Funken folgendermaßen: "Außerdem wird übersehen, daß die formulierten Anforderungen oft widersprüchlich und ungenau sind. Benutzer und Entwickler argumentieren aneinander vorbei und benutzen keine gemeinsame Sprache. Zudem existiert in den seltensten Fällen eine explizit thematisierte, konsensuale Situationsdefinition (Pasch 1994),

vielmehr interpretiert jeder die konkrete Arbeitssituation aus seiner Perspektive." (Funken, 2001, S. 31)

Es ist also eine Herausforderung, die Anforderungen in der „fremden“ Sprache einer anderen Disziplin zu erheben und in für das Software Engineering brauchbare Modelle zu übertragen. Ferner ist Studierenden die Bedeutung von Anforderungen für den folgenden Entwicklungsprozess häufig nicht klar. Primäres Ziel dieser Lehrveranstaltung ist es also, Problembewusstsein für den Stellenwert von Anforderungen und deren Erhebung zu schaffen. Durch die Zielanalyse wurde klar, dass es in dieser Lehrveranstaltung nicht primär um rein fachliche Inhalte gehen sollte, sondern um das Training überfachlicher, interdisziplinärer Kompetenzen im Kontext von Anforderungserhebung. Mit dem ursprünglichen Lehrkonzept wurde dieses Lehrziel nur teilweise erreicht. Die explizite Zieldefinition und ein Abgleich mit den tatsächlichen Inhalten der Lehrveranstaltung führten zu der Erkenntnis, dass eine auf stärkere Problemorientierung ausgerichtete Umstrukturierung und Neugewichtung der Inhalte zielführend war. In der anschließenden Umsetzungsplanung wurden vermehrt aktivierende Aufgaben eingeführt, die die Studierenden nun selbst bearbeiten müssen. Der klassische Vorlesungsstil, der bis dahin – wie häufig im Ingenieursstudium (vgl. Prince & Felder, 2006) – favorisiert war, wurde reduziert. Im Zuge der zunehmenden Aktivierung der Studierenden wurden die Aufgaben umfangreicher und die Studierenden somit an mehr Selbständigkeit herangeführt. Das Verständnis für das Thema wurde durch ein einziges durchgängiges Bespiel verstärkt, das mehrere inhaltlich voneinander unabhängige Beispiele ablöste.

4.3.3.3 Software Modellierung und -Architektur, Teil Modellierung, zweite Überarbeitungsphase

In Sedelmaier und Landes (2014a) (Anhang 1.1) ist die zweite Überarbeitungsphase und eine Fortführung des Lehrkonzepts beschrieben. Nun wurden verstärkt auf methodischer Ebene Anpassungen vorgenommen und in nahezu jede Lehreinheit aktivierende Elemente eingebaut. Um die Herausforderungen in der Anforderungserhebung und -analyse für die Studierenden noch plastischer zu machen, wurde ein „echter“ fachfremder Kunde für ein Softwareprojekt hinzugezogen. Von diesem sollten die Studierenden in Gruppen Anforderungen an ein potentielles Softwaresystem erheben. Dazu erhielten sie ein Übungsblatt mit Hinweisen zur Vorbereitung eines solchen Gesprächs, worauf bei einem solchen Kundengespräch zu achten und worüber im Vorfeld zu entscheiden ist. Hier wurde deutlich, dass die Lehrenden die Studierenden noch deutlich überschätzt hatten, denn das Kundengespräch lieferte den Studierenden kaum brauchbare inhaltliche Ergebnisse.

Das war auf fehlende Arbeits- und Selbstorganisationstechniken zurückzuführen, von denen die Lehrenden ausgingen, dass die Studierenden diese bereits mitbringen. Auch wenn die Studierenden mit der Situation deutlich überfordert waren, so zeigte sich dennoch ein deutlicher Lernerfolg hinsichtlich des Problembewusstseins. Dieser wird in einer Evaluation des Lehrkonzepts mit SECAT deutlich (Abb. 11).

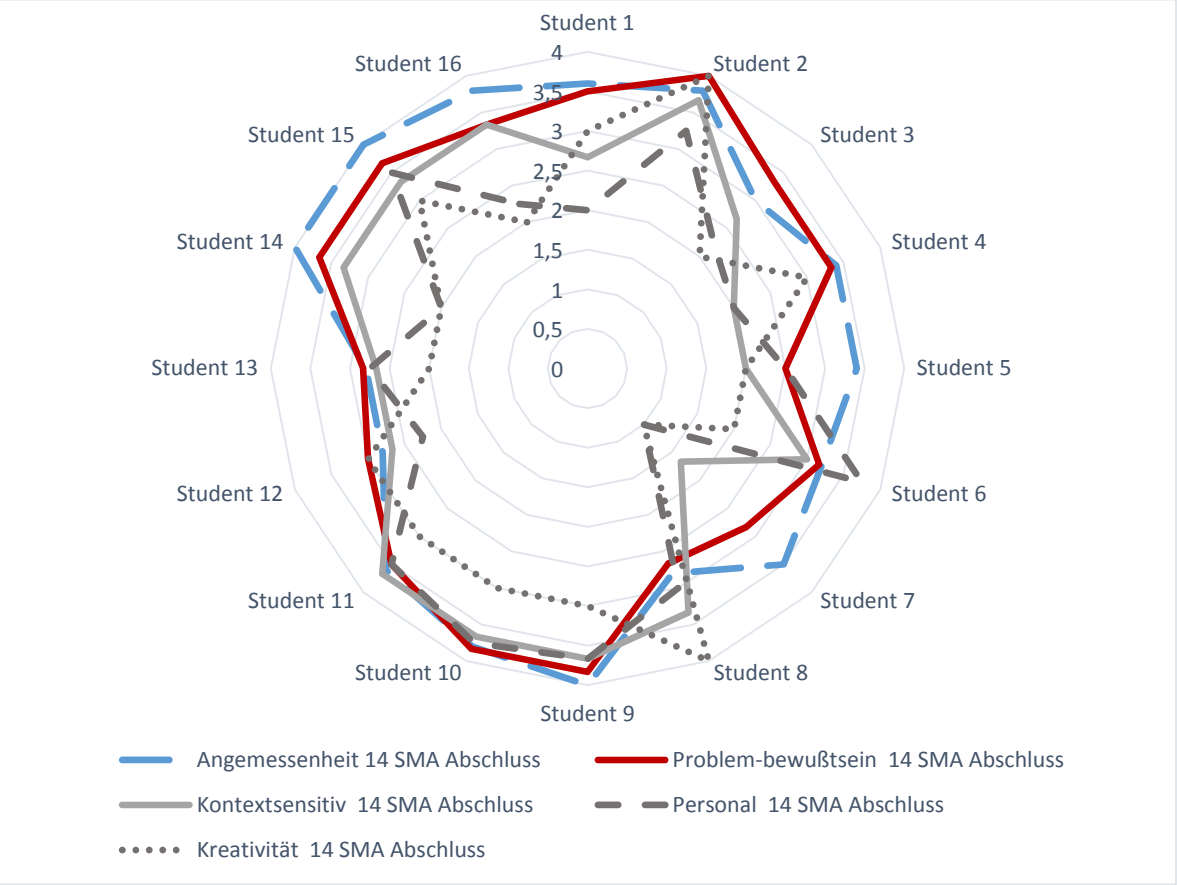


Abbildung 11 Kompetenzzuwachs in den adressierten Kriterien je Studierender

4.3.3.4 Software Modellierung und -Architektur, Teil Modellierung, dritte Überarbeitungsphase

Aufgrund der Erfahrungen aus den beiden vorherigen Durchgängen und der vorliegenden Evaluationsergebnisse mit SECAT wurden in der dritten Überarbeitungsphase sowohl die Vor- als auch die Nachbereitung eines Kundengesprächs intensiviert und systematisch mit Methoden hinterlegt. So ermöglichten etwa ein Perspektivwechsel, gezieltere Leitfragen bei der Vorbereitung des Kundentermins und umfangreiches Feedback in der Nachbereitung, das u.a. durch eine Videoaufzeichnung unterstützt wurde, den Studierenden, die erforderlichen kontextsensitiven überfachlichen Kompetenzen für Anforderungsanalyse und -erhebung zu trainieren.

Ein Vergleich der SECAT-Evaluationsergebnisse mit identischen Fragebögen für die Lehrveranstaltungen in 2014 und 2015 zeigt, dass die Weiterentwicklung des Lehr-Lern-Konzepts zu einem deutlich höheren Kompetenzzuwachs in 2015 geführt hat (Abb. 12 & 13). Daraus kann geschlossen werden, dass das Lehr-Lern-Konzept weiter verbessert wurde und nun die Studierenden die Lehrziele noch besser erreichen.

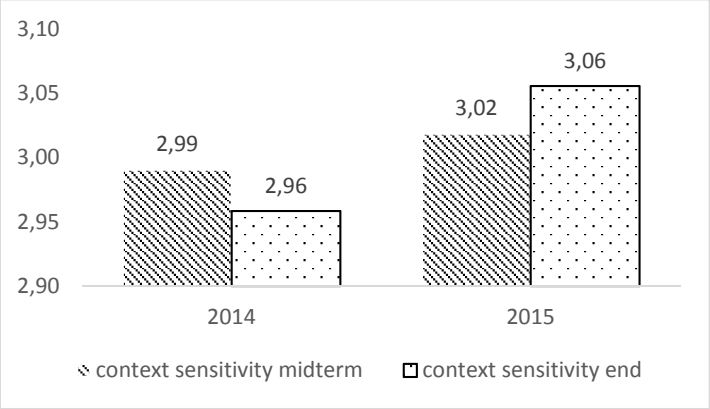


Abbildung 12 Vergleich des Kompetenzzuwachses aufgrund der Lehrveranstaltung Software-Modellierung und -Architektur in 2014 und 2015 – Kriterium Kontextsensitivität

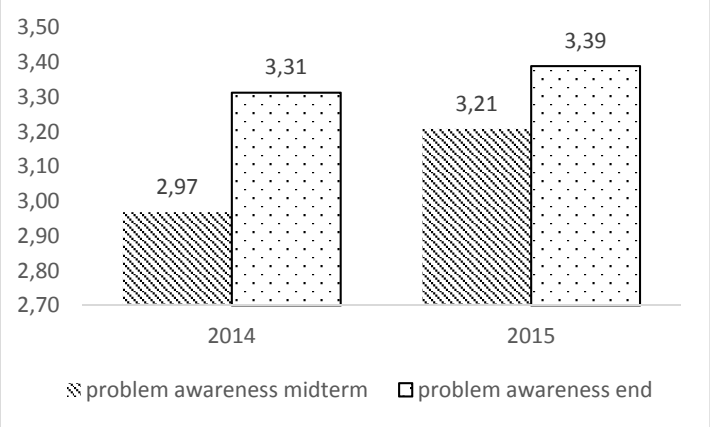


Abbildung 13 Vergleich des Kompetenzzuwachses aufgrund der Lehrveranstaltung Software-Modellierung und -Architektur in 2014 und 2015 – Kriterium Problembewusstsein

4.3.3.5 Software-Engineering-Projekt

Ein weiteres Lehrkonzept, das im Rahmen dieses Vorhabens weiterentwickelt wurde, ist das Software-Engineering-Projekt (vgl. Sedelmaier & Landes, 2014c), in das auch echte, fachfremde Kunden eingebunden sind. Die Analyse des Lehrkonzepts in Verbindung mit dem Abgleich der Lehrziele zeigte deutlich, dass die Teams, die zu Beginn ausschließlich aus Bachelorstudierenden bestanden (vgl. Landes, Sedelmaier, Pfeiffer, Mottok & Hagel, 2012), mit der zusätzlichen Teamleitung und -organisation überfordert waren. Dadurch war eine Konzentration auf die eigentlichen Lehrziele nicht

in dem von den Lehrenden gewünschten Umfang möglich. Daher wurde das Lehrkonzept dahingehend verändert, dass in jedes Bachelorteam jeweils ein Masterstudierender als Teamleiter integriert wird und sich die Lehrziele für die Bachelor- und Masterstudierenden deutlicher voneinander abgrenzen. Diese klarere Rollenverteilung ermöglicht den Studierenden, sich besser auf die Erreichung der für sie vorgesehenen Lehrziele zu konzentrieren. Verstärkt wird der Lerneffekt zusätzlich dadurch, dass die Studierenden zum Ende des Projekts ihre Zusammenarbeit im Team in einer schriftlichen Selbstreflexion bewerten. Zudem sollen sie über weitere überfachliche kontextsensitive Kompetenzen im Software Engineering reflektieren, die in diesem Projekt besonders zum Tragen kommen (Sedelmaier & Landes, 2014c). Diese Selbstreflexion wird dann in einem gemeinsamen Abschlussgespräch des Teams, einem sogenannten Post-Mortem-Review, thematisiert.

4.3.4 Beobachtungen und Erkenntnisse

Obwohl die neuen Lehrkonzepte, die deutlich von einer klassischen Vorlesung abweichen, sehr behutsam entwickelt und eingeführt wurden, stellten sie für die Studierenden etwas Neues und Ungewohntes dar. Daraus resultierten zu Beginn häufig Unsicherheit und gelegentlich auch Abwehrhaltungen den „neuen“ Methoden gegenüber. Einige forderten aktiv wieder den „Kinomodus“ ein, in dem der Lehrende Wissen präsentiert, das rezeptiv gelernt und in der schriftlichen Prüfung aufgeschrieben werden kann. Kompetenzorientierte Lehre war nur schwer als plausibler Grund anzuführen. Zunächst schien es, als ob auch Erläuterungen der Lehrziele zu Beginn des Semesters sowie auch die gezielte Begründung einzelner Methoden nicht zu einem Kulturwandel bei den Studierenden führen würden. Zu groß schienen die Berührungsängste mit Kommilitonen und erst recht mit fremden Disziplinen wie Pädagogik. Nach einigen Wochen entspannte sich die Situation dann aber sogar soweit, dass diejenigen, die in der Pflichtveranstaltung Software Engineering im dritten Semester noch nach klassischen Vorlesungen gerufen hatten, sich aktiv für das Wahlpflichtfach Software-Modellierung und -Architektur im vierten Semester eingeschrieben haben. Die Drop-out-Quote während des Semester war sehr gering, vielleicht auch, weil zwischen jedem Studierenden und den Lehrenden zu Semesterbeginn eine Art Teilnehmervertrag unterzeichnet wurde, in dem man sich explizit auf „neue“ kompetenzorientierte Lehr-Lern-Formen verständigte.

Eine weitere Beobachtung war, dass es vielen Studierenden an Arbeitstechniken und Fähigkeiten zur Selbstorganisation fehlte. Sie standen manchen vermeintlich präzise formulierten Arbeitsaufträgen hilf- und planlos gegenüber. Diese Erkenntnis führte dazu, den Arbeitsbereich über Software Engineering hinaus auszuweiten und bereits in

einer Grundlagenveranstaltung im ersten Semester immer wieder aktivierende Elemente einzustreuen, die die Studierenden langsam an Arbeitstechniken und grundlegende Fähigkeiten wie Analyse- und Abstraktionsvermögen heranführen. Sie sollen von Beginn an eine aktivere Rolle im Lernprozess übernehmen, anstatt nur passiv einer Vorlesung zu lauschen.

So fordern die Lehrenden in der Pflichtveranstaltung „Grundlagen der Informatik“ im ersten Semester nach jedem Themenbereich der Veranstaltung ein sogenanntes Ten-Minute-Paper von den Studierenden ein, in dem sie auf einer halben Seite bis einer Seite die für sie wesentlichen Erkenntnisse zum Thema zusammenfassen sollen. Dieses Papier wird noch in der Lehrveranstaltung einem Peer-Review unterzogen. So erhalten die Studierenden Kommentare und Ergänzungen zu ihren Zusammenfassungen und können gleichzeitig möglicherweise vergessene Aspekte bei ihren Kommilitonen entdecken. Dann sammeln die Lehrenden die Papiere ein, um sich selbst ein Bild über den Erkenntnisstand der Studierenden zu machen und ggf. einzelne Aspekte des Themas nochmals zu wiederholen oder zu vertiefen. Am Ende des Semesters erhält jeder Studierende seine Ten-Minute-Papers als Art Überblick über das Fach zurück.

In den höheren Semestern, besonders im Wahlpflichtfach Software-Modellierung und -Architektur, in dem verstärkt kompetenzorientierte Lehrmethoden zum Einsatz kommen, erhalten die Studierenden mit jedem Arbeitsauftrag eine kurze schriftliche Erläuterung, wozu der jeweilige Arbeitsauftrag dienen soll und welche Kompetenzen er adressiert. Diese Kommunikation von Lehrzielen erleichtert es den Studierenden, den Sinn und das Ziel der jeweiligen Aufgabenstellung zu verstehen.

Schritt für Schritt wurde so das Curriculum für den Bachelorstudiengang Informatik mit kompetenzorientierten Elementen angereichert, indem es vom ersten Semester an zunächst allgemeinere überfachliche Kompetenzen wie Arbeitstechniken, im späteren Studienverlauf zunehmend kontextsensitive überfachliche Kompetenzen im Software Engineering adressiert, die immer im Kontext der fachlichen Disziplin adressiert werden.

Insgesamt lässt sich feststellen, dass sich mit zunehmender Einführung kompetenzorientierter Lehrformen im Software Engineering auch die Prüfungsnoten verbessert haben. Als Beispiel kann hier die Pflichtveranstaltung Software Engineering angeführt werden. So erzielten Informatikstudierende der Hochschule Coburg im dritten Semester aufgrund des neuen Lehrkonzepts bessere Prüfungsergebnisse als ihre Kommilitonen in den Vorjahren: neben einer deutlichen Verbesserung der durchschnittlichen erreichten Punktezahl in der Prüfung ist insbesondere die Durchfallquote signifikant gesunken, seit im Wintersemester 2014/2015 ein

grundlegender Umbau der Lehrveranstaltung in Richtung aktivierender und induktiver Lehrkonzepte erfolgt ist. So ist die Durchfallquote von ca. 20 Prozent in den Vorjahren auf knapp acht Prozent im Wintersemester 2014/15 gefallen (Abb. 14). Dies kann als Indiz dafür gesehen werden, dass die neuen Lehrformen die Qualität der Lehre für eine bedeutende Anzahl von Studierenden verbessern.

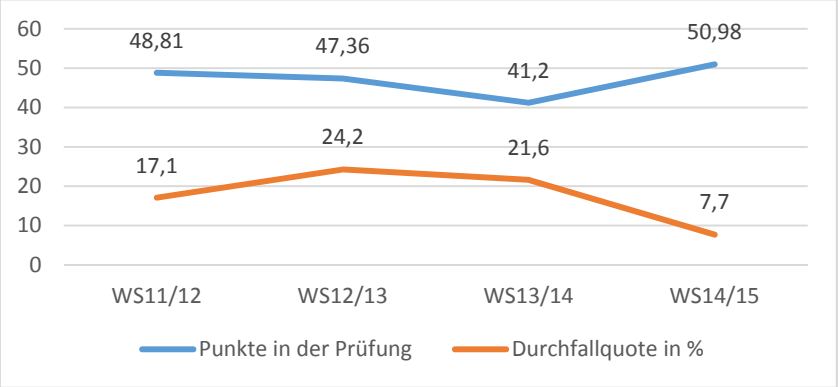


Abbildung 14 Entwicklung Prüfungserfolg und Durchfallquote Lehrveranstaltung "Software Engineering" in Coburg

5 Interdisziplinäre Hochschul-Fachdidaktik für Software Engineering

Ziel der Hochschul-Ausbildung im Software Engineering ist es, Studierende für den späteren Berufsalltag zu qualifizieren. Dazu benötigen sie neben solidem Fachwissen vor allem anwendbares Handlungswissen. Diese Kompetenzen, die sich aus fachlichen und überfachlichen zusammensetzen, gilt es im Studium zu adressieren.

Zentrale Herausforderungen in der Lehre von Software Engineering sind wie bereits in Abschnitt 2.1 erläutert zum einen, das Problembewusstsein für die Komplexität von Software Engineering sowie ein Verständnis für die Bedeutung von Interdisziplinarität zu schaffen, und zum anderen die dafür erforderlichen Kompetenzen gezielt zu adressieren.

5.1 Schaffung von Problembewusstsein und Verständnis für Interdisziplinarität

Eine der zentralen Herausforderungen in der Lehre von Software Engineering ist es, bei den Studierenden Problembewusstsein für die zu lernenden Themen und Kompetenzen zu wecken. Problembewusstsein stellt zudem auf inhaltlicher Ebene eine zentrale Kompetenz im Software Engineering dar und ist in SWEBOS näher beschrieben.

In der Hochschulausbildung gilt es, zuerst ein Bewusstsein für den Sinn der zu lernenden Methoden und Techniken des Software Engineering bei den Studierenden zu schaffen. Dies schließt ein, dass die Studierenden die Herausforderungen vor allem in der Phase der Anforderungserhebung und -analyse, in der die Interdisziplinarität besonders deutlich zu Tage tritt, erkennen und bewältigen lernen. Dazu gilt es, zuerst ein (fachfremdes) Problem zu erleben und zu erfassen, bevor eine mögliche Lösung geliefert wird. Aufgrund der Komplexität des Fachs Software Engineering fehlen Studierenden andernfalls häufig das Verständnis und der Nutzen des Lerninhalts, was somit auch zu einem schlechteren Lernergebnis führt. Es scheint für Studierende sinnlos, sich mit Lösungen für Probleme zu beschäftigen, die sie selbst noch nie erfahren haben. In klassischen Software Engineering Vorlesungen wurden bis vor wenigen Jahren theoretische Lösungen für Probleme gelehrt, die die Studierenden selbst noch nie hatten und nicht einmal erahnen können. Ein Schlüssel in der Ausbildung von Software Engineering scheint also zu sein, zunächst das Problem greifbar zu machen, um so ein Problemverständnis zu ermöglichen (vgl. Glasersfeld, 2001, S. 169). In Anlehnung an Holzkamp kann auch besonders im Software Engineering davon ausgegangen werden, dass Lernen dann erst möglich wird, „wenn das Subjekt in seinem normalen Handlungsvollzug auf Hindernisse oder Widerstände

gestoßen ist und sich dabei vor einer ‚Handlungsproblematik' sieht, die es nicht mit den aktuell verfügbaren Mitteln und Fähigkeiten, sondern nur durch den Zwischenschritt oder (produktiven) Umweg des Einschaltens einer ‚Lernschleife' überwinden kann." (Holzkamp, 1996, S. 21) Nur so eröffnen sich im Sinne einer konstruktivistischen Grundhaltung auch die Möglichkeiten, Methoden und Techniken des Software Engineering im interdisziplinären Kontext zu lernen. "Teaching should not begin with the presentation of sacred truths, but rather by creating opportunities for making the students think." (Glasersfeld, 2001, S. 171)

Um dieses Ziel zu erreichen, scheinen problembasierte bzw. problemorientierte Lehrkonzepte sinnvoll (vgl. Savin-Baden, 2001). Diese lassen sich im Software Engineering auf verschiedene Weise umsetzen. So hilft es Studierenden etwa, wenn wie in Sedelmaier und Landes (2015b) (Anhang 1.4) beschrieben zu Semesterbeginn ein Beispiel aus dem Berufsalltag für die Thematik sensibilisiert. Auch Siebert stellt fest: "Lerngegenstände, die nicht an vorhandene kognitive Systeme angekoppelt werden können, hängen gleichsam 'in der Luft' und werden meist schnell vergessen." (1997, S. 105) Diese Anknüpfung kann etwa eine Aufgabenstellung für die Studierenden sein, in der sie für einen möglichen Kunden ein Angebot für ein Softwaresystem erstellen sollen. So überlegen sie sich zuerst, was die nächsten Schritte zur Angebotserstellung sein könnten, was sie dazu wissen müssen oder benötigen und wo mögliche Schwierigkeiten auftauchen könnten. Das Problembewusstsein für die zu lernenden Methoden wird noch verstärkt, wenn dieses initiale Beispiel während des Semesters stetig wieder aufgenommen und ausgebaut und daran die Komplexität des Themas verdeutlicht wird. Dadurch wird es den Studierenden erleichtert, die Auswirkungen zu erkennen, wenn Methoden und Techniken des Software Engineering nicht angewandt werden. So gewinnen im Verlauf des Lernprozesses die Methoden für die Studierenden zunehmend an Bedeutung, da die Studierenden deren Nutzen erkennen.

Der Lerneffekt wird zudem deutlich gesteigert (siehe Abschnitt 4.3.4), wenn möglichst reale Situationen in die Lehrkonzepte eingebaut werden, in denen auch die Zusammenhänge zwischen den adressierten Kompetenzen erkennbar werden. So bringt etwa ein Kundengespräch, das nicht so verläuft, wie es sich die Studierenden theoretisch vorgestellt haben, die Einsicht, dass selbst bei optimaler Vorbereitung unvorhersehbare und unkontrollierbare Ereignisse zu einem unplanbaren Verlauf des Gesprächs führen können. Die Rückmeldungen der Studierenden zeigen immer wieder, dass in Situationen, in denen sie spontan reagieren müssen, nicht nur ein Bewusstsein für die zu lernenden Techniken und Methoden sowie die Notwendigkeit interdisziplinärer, überfachlicher Kompetenzen wächst, sondern darüber hinaus auch ein hoher Trainingseffekt vor allem bzgl. überfachlicher Kompetenzen auftritt.

Gerade das Verständnis für komplexe Zusammenhänge und die Bedeutung der Interdisziplinarität kann nicht losgelöst vom Kontext gelernt werden. Das heißt also für die Lehre von Software Engineering, dass versucht werden sollte, die Komplexität von Software Engineering möglichst realitätsnah in der Lehre abzubilden, auch wenn das meist nur in Ausschnitten möglich ist.

Eine weitere Möglichkeit, die Studierenden für die Interdisziplinarität und die Notwendigkeit von überfachlichen Kompetenzen im Software Engineering zu sensibilisieren, ist beispielsweise, sie immer wieder auch die Sichtweise der Kunden einnehmen zu lassen, bevor daraus Handlungsmöglichkeiten für das eigene Verhalten abgeleitet werden. So werden Studierende etwa zur Vorbereitung auf ein reales Kundengespräch gefragt, wie sich ein Vertreter eines Softwareanbieters verhalten bzw. das Gespräch verlaufen müsste, damit sie selbst als Kunden einen Auftrag an eben diese Softwarefirma vergeben würden. Die Studierenden erarbeiten sich so selbst, worauf sie achten würden, wenn sie der Kunde wären. Diese Erkenntnisse lassen sich dann leichter auf die Vorbereitung eines eigenen Kundengesprächs übertragen als der theoretisch erhobene Zeigefinger des Lehrenden. Dieses Vorgehen schärft den Blick für mögliche Probleme und somit auch das Problembewusstsein.

Es lässt sich also zusammenfassen, dass problembasierte Lehrkonzepte mit möglichst realistischen und durchgängigen Beispielen und Perspektivwechseln sowie real erlebbaren Situationen aus dem Software Entwicklungsprozess helfen, das Verständnis für die Methoden und Techniken des Software Engineering sowie die damit verbundene Notwendigkeit überfachlicher Kompetenzen zu schärfen.

Diese problembasieren Ansätze können eher einem induktiven Grundverständnis von Lehre zugerechnet werden. Für die Lehre von Software Engineering scheinen also auch induktive Konzepte erfolgversprechender zu sein. Induktive Lehrkonzepte gehen im Gegensatz zu deduktiven von einem Problem im Besonderen aus, um die gewonnenen Erkenntnisse dann zu abstrahieren und auf allgemeinere Sachverhalte zu übertragen.

5.2 Kompetenztraining in der Hochschullehre von Software Engineering

Nachdem die Studierenden die Notwendigkeit für Software Engineering und die damit verbundenen interdisziplinären und überfachlichen Kompetenzen erkannt haben, gilt es nun, die Kompetenzen zu adressieren, die im Software Engineering notwendig sind.

Um diese Kompetenzen zu adressieren, scheinen klassische Vorlesungen eher ungeeignet, da ihnen ein reproduktionsorientierter Lernstil zugrunde liegt (vgl. Konrad, 2014, S. 101). Eine klassische Vorlesung folgt einer linearen Verarbeitungsstrategie,

die durch wörtliches Wiederholen und Abrufen sowie durch "oberflächliches" Verarbeiten von Information (Auswendiglernen, Wiedergeben) charakterisiert ist. Die Steuerung des Lernprozesses erfolgt von außen, indem Wissen, das von außen vorgegeben wird, wörtlich aufgenommen wird. Eine klassische schriftliche Prüfung überprüft und bescheinigt den Lernerfolg.

Im Gegensatz dazu führt im Software Engineering eher ein bedeutungsorientierter Lernstil zum Erfolg. Hier werden Informationen "tief" verarbeitet (Verstehen und Einsicht) und die Anwendung von Tiefenverarbeitungsstrategien (In-Beziehung-setzen, Strukturieren, kritische Stellungnahme) tritt in den Vordergrund. Besonders überfachliche Kompetenzen lassen sich nicht theoretisch lernen. Zuhören oder Zusehen, wie jemand einen Sachverhalt strukturiert, führt nicht zwangsläufig dazu, dass diese Kompetenz in der Art gefördert wird, dass der Zuhörende oder Zusehende danach selbst einen fremden Sachverhalt strukturieren kann (vgl. Dewey, 2011, S. 106). Auch Erpenbeck & Sauter plädieren dafür, Kompetenzen in „neuartigen, offenen und realen Problemsituationen kreativ handelnd“ zu trainieren, um neue, unbekannte Lösungswege abzuleiten (2013, S. 32-33). Aus der gegebenen Problemstellung in Verbindung mit den vorliegenden Ergebnissen kann geschlossen werden, dass im Software Engineering aktivierende Lehr-Lernformen erfolgreicher sind. Diese ermöglichen es den Studierenden, Kompetenzen zu trainieren und situationsbezogen im jeweiligen Kontext anzuwenden und Methoden und Techniken anzupassen. Für aktivierende Lehre sind zahlreiche Methoden zu finden (vgl. z. B. Reich, 2012), insbesondere seit das Paradigma des Konstruktivismus zur aktuellen Strömung in der Didaktik geworden ist (vgl. Siebert, 1999, S. 20). Aktivierende Lehrformen setzen sich aus mehreren Elementen zusammen, und ermöglichen den Studierenden so, komplexe Kompetenzen zu trainieren. Prince (2004) weist Erfolge aktivierenden Lehrmethoden für diese Lehrziele aus. Selbstverständlich müssen diese Methoden gemäß der in Abschnitt 4.3.2 erläuterten Vorgehensweise systematisch und zielorientiert ausgewählt, kombiniert und eingesetzt werden.

Zusammenfassend lässt sich festhalten, dass aktivierende Lehrformen sehr gut geeignet sind, um Kompetenzen im Software Engineering zu adressieren, da Studierende mögliche Lösungen und Methoden selbständig und direkt anwenden können müssen – genau dies ist bei aktivierenden Lehrformen ein zentrales Merkmal.

6 Zusammenfassung und Ausblick

Software zieht sich mittlerweile durch alle Bereiche des täglichen Lebens. Mit der Bedeutung von Software wächst auch die Bedeutung der Ausbildung derer, die diese Software trotz aller Komplexität qualitativ hochwertig entwickeln können. Jedoch fehlten bisher methodisch fundierte Aussagen, was alles zu einer guten Software-Engineering-Ausbildung gehört.

Diese Arbeit entwirft ein Forschungsdesign, das zu einer interdisziplinären Fachdidaktik für Software Engineering führt. Ausgehend von Kompetenzprofilen *guter* Software-Ingenieure werden dabei Maßnahmen abgeleitet, um diese Kompetenzziele zu erreichen. Diese Maßnahmen sind schließlich kompetenzorientiert zu evaluieren.

Kernelemente des Forschungsansatzes umfassen also ein Kompetenzprofil, die darauf aufbauende kompetenzorientierte Weiterentwicklung von Lehrkonzepten sowie die systematische Evaluation der Wirksamkeit dieser Lehrkonzepte, um daraus Rückschlüsse auf die Lehr-Lern-Prozesse und ihre Wirkzusammenhänge zu ziehen.

Das Ergebnis der gewonnenen Erkenntnisse ist ein Vorschlag für eine interdisziplinäre Fachdidaktik für Software Engineering. Dabei orientiert sie sich, wie in Abschnitt 2.1 erläutert, an den Aufgaben einer Fachdidaktik (vgl. Kron, 2008, S. 29; Plöger, 1999, S. 17).

Mit dem Software Engineering Body of Skills (SWEBOS) (vgl. Abschnitt 4.1) wurde ein wissenschaftlich fundiertes Kompetenzprofil erarbeitet. In Verbindung mit fachlichen Richtlinien wie etwa dem SWEBOK (vgl. Bourque & Fairley, 2014) präzisiert dieses Kompetenzprofil die Lehrziele der Software Engineering Ausbildung und bietet eine Orientierung für die Auswahl der Lehrziele.

Im Zuge dieser Arbeit wurden die zentralen Herausforderungen für die Lehre von Software Engineering herausgearbeitet. Dabei handelt es sich zum einen um das mangelnde Problembewusstsein der Studierenden im Hinblick auf die Notwendigkeit des Fachs für ihre spätere berufliche Praxis und die Bedeutung interdisziplinärer Kompetenzen. Zum anderen fehlt Studierenden das Verständnis für komplexe Zusammenhänge im Software Engineering. Zudem sind in der Lehre neben Fachwissen zahlreiche damit eng verzahnte überfachliche, auch interdisziplinäre Kompetenzen zu adressieren.

Die vorliegende Arbeit entwickelt ein Vorgehen, um Lehrveranstaltungen ziel- und kompetenzorientiert weiterzuentwickeln und deren Wirksamkeit mit Hilfe des Kompetenzbewertungsansatzes SECAT zu evaluieren. Daraus werden didaktische Konzepte abgeleitet, mit deren Hilfe möglichst viele Lehrziele erreicht werden können.

Die Umsetzung dieser Konzepte legt den Schluss nahe, dass aktivierende und induktive Lehrformen den Kompetenzaufbau im Software Engineering am besten unterstützen.

In weiteren Entwicklungsschritten gilt es nun, die Curricula der Hochschulstudiengänge, die Software Engineering enthalten, kompetenzorientiert weiterzuentwickeln. Dies schließt eine Anpassung der Prüfungsformen und -formate ein. Hierzu kann SECAT einen Grundstein legen.

In weiteren Untersuchungen ist das Bewertungsinstrument SECAT zu validieren, das Kompetenzprofil SWEBOS weiterzuentwickeln und noch stärker mit SECAT zu verzahnen.

7 Literaturverzeichnis

Achtenhagen, F. (2006). Lehr-Lern-Forschung. In R. Arnold & A. Lipsmeier (Hrsg.), *Handbuch der Berufsbildung* (S. 586–609). Wiesbaden: VS Verlag für Sozialwissenschaften.

Artelt, C., Weinert, S. & Carstensen, C. H. (2013). Assessing competencies across the lifespan within the German National Educational Panel Study (NEPS). Editorial. *jero, 5*(2), 5–14.

Barre, K. (2012). "Bildung" und "Widerstand" im Kompetenzparadigma. In K. Barre, C. Hahn, P. Dehnbostel, A. Jastrzebski, A. Flasinska & K. Büchter (Hrsg.), *Kompetenz. Fragen an eine (berufs-)pädagogische Kategorie* (Berufsbildung, Bd. 2, S. 93–126). Hamburg: Helmut-Schmidt-Univ. Fak. für Geistes- und Sozialwiss.

Bender, W., Lerch, S. & Scheffel, M. *Interdisziplinäre Kompetenzen Studierender evaluieren. 2. Zwischenbericht der Wissenschaftlichen Begleitstudie zum Projekt "Der Coburger Weg" zum 31.05.2014*.

Bernard, F. (2001). Anforderungen des Lernfeldkonzepts an die technikdidaktische Ausbildung. *Die berufsbildende Schule, 53*(10), 299–305.

Biehler, R. & Leuders, T. (2014). Kompetenzmodellierungen für den Mathematikunterricht – Eine Zwischenbilanz aus Sicht der Mathematikdidaktik. *J Math Didakt, 35*(1), 1–5.

Bleher, W. (2001). *Das Methodenrepertoire von Lehrerinnen und Lehrern des Faches Technik. Eine empirische Untersuchung an Hauptschulen in Baden-Württemberg* (Didaktik in Forschung und Praxis, Bd. 3). Hamburg: Kovac.

Bloch, M., Blumberg, S. & Laartz, J. (2012). *Delivering large-scale IT projects on time, on budget, and on value.* Zugriff am 21.10.2015. Verfügbar unter http://www.mckinsey.com/insights/business_technology/delivering_large-scale_it_projects_on_time_on_budget_and_on_value.

Blömeke, S., Kaiser, G. & Lehmann, R. (2011). Messung professioneller Kompetenz angehender Lehrkräfte. "Mathematics Teaching in the 21st Centrury" und die IEA-Studie TEDS-M. In H. Bayrhuber, U. Harms, B. Muszynski, B. Ralle, M. Rothgangel, L.-H. Schön et al. (Hrsg.), *Empirische Fundierung in den Fachdidaktiken* (Fachdidaktische Forschungen, Bd. 1, S. 9–25). Münster, New York, NY, München, Berlin: Waxmann.

Bloom, B. S. (1972). *Taxonomie von Lernzielen im kognitiven Bereich* (Beltz Studienbuch). Weinheim: Beltz.

Blossfeld, H.-P., Roßbach, H.-G. & Maurice, J. von (Hrsg.) (2011) Education as a Lifelong Process [Themenheft], *14*(S2).

Böttcher, A., Schlierkamp, K., Thurner, V. & Zehetmeier, D. (2015). Software Engineering Project Simulation in Student Entry Phase of Computer Scientists-to-be. In IEEE (Hrsg.), *Global Engineering Education Conference (EDUCON)* (S. 486–493). IEEE.

Braun, E. (2008). *Das Berliner Evaluationsinstrument für selbsteingeschätzte studentische Kompetenzen (BEvaKomp)*. Göttingen: V & R Unipress.

Braun, E. & Hannover, B. (2008). Kompetenzmessung und Evaluation von Studienerfolg. In N. Jude, J. Hartig & E. Klieme (Hrsg.), *Kompetenzerfassung in pädagogischen Handlungsfeldern - Theorien, Konzepte und Methoden*

(Bildungsforschung, Bd. 26, S. 153–160). Bonn, Berlin: Bundesministerium für Bildung und Forschung (BMBF).

Bretschneider, M., Hummesheim, S., Neß, H., Reimer, M., Seidel, S., Seusing, B. et al. (2007). *Weiterbildungspass mit Zertifizierung informellen Lernens" (ProfilPASS). BLK-Verbundprojekt; Endbericht der Erprobungs- und Evaluationsphase*. Frankfurt (am Main): DIE.

Brychcy, U., Büschemann, K.-H. & Rubner, J. (2003, 15. Oktober). Toll Collect - Technisch machbar — aber noch nicht jetzt - Süddeutsche.de. *Süddeutsche Zeitung.* Zugriff am 19.08.2015. Verfügbar unter http://www.sueddeutsche.de/wirtschaft/toll-collect-technisch-machbar-aber-noch-nicht-jetzt-1.906381.

Dehnbostel, P., Neß, H. & Overwien, B. (2009). *Der Deutsche Qualifikationsrahmen. (DQR) ; Positionen, Reflexionen und Optionen ; Gutachten im Auftrag der Max-Traeger-Stiftung* (Die GEW diskutiert). Frankfurt (Main): GEW Hauptvorstand.

Deitel, P. (2012). *Java. How to Program: International Version* (9. Aufl.). Harlow: Pearson Education Limited.

Arbeitskreis Deutscher Qualifikationsrahmen (AK DQR) (Hrsg.). (2011). *Deutscher Qualifikationsrahmen für lebenslanges Lernen,* Bundesministerium für Bildung und Forschung (BMBF); Ständige Konferenz der Kultusminister der Länder. Zugriff am 27.07.2014. Verfügbar unter http://www.dqr.de/media/content/Der_Deutsche_Qualifikationsrahmen_fue_lebenslanges_Lernen.pdf.

Deutsches Institut für Erwachsenenbildung (2012). *ProfilPASS. Stärken kennen, Stärken nutzen ; plus ePortfolio* (2. Aufl.). Bielefeld: Bertelsmann.

Dewey, J. (2011). *Democracy and Education: An Introduction to the Philosophy of Education:* IndoEuropean.

Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM, 15*(10), 859–866.

Emam, K. E. & Koru, A. G. (2008). A Replicated Survey of IT Software Project Failures. *IEEE Softw., 25*(5), 84–90.

Empfehlungen zur Qualitätsverbesserung von Lehre und Studium (2008). Köln: WR Wissenschaftsrat.

Erpenbeck, J. (2012). *Der Königsweg zur Kompetenz. Grundlagen qualitativ-quantitativer Kompetenzerfassung* (Kompetenzmanagement in der Praxis, Bd. 6). Münster, New York, NY, München, Berlin: Waxmann.

Erpenbeck, J. & Sauter, W. (2013). *So werden wir lernen! Kompetenzentwicklung in einer Welt fühlender Computer, kluger Wolken und sinnsuchender Netze*. Berlin, Heidelberg: Springer Gabler.

Eveleens, J. L. & Verhoef, C. (2010). The rise and fall of the Chaos report figures. *IEEE Softw., 27*(1), 30–36.

Fast, L. (2006). Entwicklungslinien für Fachkonzepte und Fachraumkonzepte für Technikunterricht. *Unterricht. Arbeit + Technik*. (30), 44–46.

Funken, C. (2001). *Die Modellierung der Welt. Wissenssoziologische Studien zur Software-Entwicklung*. Opladen: Leske + Budrich.

Geertz, C. (1987). *Dichte Beschreibung. Beiträge zum Verstehen kultureller Systeme* (Suhrkamp-Taschenbuch Wissenschaft, 12. Aufl.). Frankfurt a.M: Suhrkamp.

Gesellschaft für Informatik e.V. (GI). (2006). *Was ist Informatik? – Unser Positionspapier.* Zugriff am 11.08.2015. Verfügbar unter http://www.gi.de/fileadmin/redaktion/Download/was-ist-informatik-lang.pdf.

Glaser, B. G. & Strauss, A. L. (2010). *Grounded Theory. Strategien qualitativer Forschung* (3. Aufl.). Bern: Huber.

Glasersfeld, E. von (2001). Radical constructivism and teaching. In C. Braslavsky (Hrsg.), *Constructivism and education. No 118* (Prospects: quarterly review of comparative education, XXXI, 2, S. 161–174). UNESCO International Bureau of Education. Zugriff am 12.11.13. Zugriff am 12.11.13. Verfügbar unter http://unesdoc.unesco.org/images/0012/001249/124945eo.pdf.

Gnahs, D. (2010). *Kompetenzen - Erwerb, Erfassung, Instrumente*. Bielefeld: Bertelsmann.

Gudjons, H. (2006). *Pädagogisches Grundwissen. Überblick, Kompendium, Studienbuch* (9. Aufl.). Bad Heilbrunn: Klinkhardt.

Bourque, P. & Fairley, R. E. (Hrsg.). (2014). *Guide to the Software Engineering Body of Knowledge Version 3.0 - SWEBOK,* IEEE Computer Society. Zugriff am 13.03.2014. Verfügbar unter http://www.computer.org/ieeecs-swebokdelivery-portlet/swebok/SWEBOKv3.pdf.

Hagel, G. & Mottok, J. (2011). Planspiel und Briefmethode für die Software Engineering Ausbildung - ein Erfahrungsbericht. In J. Ludewig & A. Böttcher (Hrsg.), *Software Engineering im Unterricht der Hochschulen" (SEUH 2011)* (S. 10–15). Zugriff am 08.07.2015. Verfügbar unter http://ceur-ws.org/Vol-695/beitrag3-hagel-mottok.pdf.

Hank, B. (2014). Vorstellungen zum Verbrennungsprozess – Entwicklung wissenschaftlicher Konzepte bei Schülerinnen und Schülern. *PAradigma*. (3), 18–29.

Hartig, J. (2008). Kompetenzerfassung von Bildungsprozessen. In N. Jude, J. Hartig & E. Klieme (Hrsg.), *Kompetenzerfassung in pädagogischen Handlungsfeldern - Theorien, Konzepte und Methoden* (Bildungsforschung, Bd. 26, S. 15–25). Bonn, Berlin: Bundesministerium für Bildung und Forschung (BMBF).

Haustein-Teßmer, O. (2003, 24. September). Software-Chaos bei Toll Collect. *Die Welt.* Zugriff am 19.08.2015. Verfügbar unter http://www.welt.de/wirtschaft/article261975/.

Heimann, P., Otto, G. & Schulz, W. (Hrsg.) (1965). *Unterricht. Analyse und Planung*. Hannover: Schroedel.

Henseler, K. (1996). *Methodik des Technikunterrichts*. Bad Heilbrunn: Klinkhardt.

Heymann, H. W. (1996). *Allgemeinbildung und Mathematik* (Reihe Pädagogik, Bd. 13, Dr. nach Typoskript). Weinheim: Beltz.

Heyse, V. (Hrsg.) (2010). *Grundstrukturen menschlicher Kompetenzen. Praxiserprobte Konzepte und Instrumente* (Kompetenzmanagement in der Praxis, Bd. 5). Münster: Waxmann.

Höhn, R., Höppner, S. & Rausch, A. (2008). *Das V-Modell XT. Anwendungen, Werkzeuge, Standards* (EXamen.press). Berlin, Heidelberg: Springer.

Holzkamp, K. (1996). Widen den Lehr-Lern-Kurzschluß. In R. Arnold (Hrsg.), *Lebendiges Lernen* (Grundlagen der Berufs- und Erwachsenenbildung, Bd. 5, S. 21–30). Baltmannsweiler: Schneider Verlag Hohengehren.

Hubwieser, P. (2007). *Didaktik der Informatik. Grundlagen, Konzepte, Beispiele* (EXamen.press, 3. Aufl.). Berlin: Springer.

Hüttner, A. (2005). *Technik unterrichten. Methoden und Unterrichtsverfahren im Technikunterricht* (2. Aufl.). Haan-Gruiten: Europa-Lehrmittel.

IEEE Computer Society. (2014). *Software Engineering Competency Model Version 1.0 (SWECOM)*.

IEEE Computer Society; Association for Computing Machinery ACM. (2004, 23. August). *Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. A Volume of the Computing Curricula Series,* IEEE Computer Society; Association for Computing Machinery ACM. Zugriff am 24.09.2013. Verfügbar unter http://sites.computer.org/ccse/SE2004Volume.pdf.

Jungwirth, H. (2014). *Beitrag zur Theoriearbeit und LehrerInnenbildung in der interpretativen mathematikdidaktischen Forschung*. Münster: Waxmann.

Jürgens, E. (1992). *Leistung und Beurteilung in der Schule. Eine Einführung in Leistungs- und Bewertungsfragen aus pädagogischer Sicht* (1. Aufl). Sankt Augustin: Academia-Verl.

Kabicher, S., Motschnig-Pitrik, R. & Figl, K. (2009). What Competences do Employers, Staff and Students expect of a Computer Science Graduate? In *39th IEEE Frontiers in Education Conference (FIE)* (S. T1A-1 - T1A-6)

Kattermann, U., Duit, R., Gropengießer, H. & Komorek, M. (1997). Das Modell der Didaktischen Rekonstruktion - Ein Rahmen für naturwissenschaftliche Forschung und Entwicklung. *Zeitschrift für Didaktik der Naturwissenschaften, 3*(3), 3–18.

Kauffeld, S. (2002). Das Kasseler-Kompetenz-Raster (KKR) - ein Beitrag zur Kompetenzmessung. In U. Clement & R. Arnold (Hrsg.), *Kompetenzentwicklung in der beruflichen Bildung* (Schriften der Deutschen Gesellschaft für Erziehungswissenschaft (DGfE), S. 131–151). Wiesbaden: VS Verlag für Sozialwissenschaften.

Kircher, E. (2015). *Physikdidaktik. Theorie und Praxis* (Springer-Lehrbuch, 3. Aufl.). Berlin: Springer Spektrum.

Klafki, W. (1964). Didaktische Analyse als Kern der Unterrichtsvorbereitung. In H. Roth & A. Blumenthal (Hrsg.), *Didaktische Analyse* (Auswahl, Grundlegende Aufsätze aus der Zeitschrift Die deutsche Schule Reihe A, Bd. 1, S. 5–34). Hannover: Schroedel.

Konrad, K. (2014). *Lernen lernen – allein und mit anderen: Konzepte, Lösungen, Beispiele:* Springer Fachmedien Wiesbaden.

Kron, F. W. (2008). *Grundwissen Didaktik* (UTB, Bd. 8073, 5. Aufl.). München, Basel: E. Reinhardt.

Landes, D., Sedelmaier, Y., Pfeiffer, V., Mottok, J. & Hagel, G. (2012). Learning and teaching software process models. In *Global Engineering Education Conference (EDUCON)* (S. 1153–1160). Verfügbar unter http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6201191.

Lerch, S. (2014). Sprechen Sie Interdisziplinär? Zur Besonderheit interdisziplinärer Kompetenzen. In C. Schier & E. Schwinger (Hrsg.), *Interdisziplinarität und Transdisziplinarität als Herausforderung akademischer Bildung. Innovative Konzepte für die Lehre an Hochschulen und Universitäten* (Pädagogik, S. 79–94). Bielefeld, Berlin: Transcript; De Gruyter.

Leuders, T. (2009). *Qualität im Mathematikunterricht in der Sekundarstufe I und II* (6. Aufl.). Berlin: Cornelsen-Scriptor.

Leuders, T. (Hrsg.) (2011). *Mathematik-Didaktik. Praxishandbuch für die Sekundarstufe I und II* (6. Aufl.). Berlin: Cornelsen-Scriptor.

Li, P. L., Ko, A. J. & Zhu, J. (2015). What Makes a Great Software Engineer? In *37th International Conference on Software Engineering (ICSE)*

Ludewig, J. & Böttcher, A. (Hrsg.) (2011). *Software Engineering im Unterricht der Hochschulen (SEUH)*.

Maaß, K. (2007). Und man braucht sie doch! Nützlichkeit von Mathematik erfahrbar machen. *Praxis der Mathematik*. (13), 1–9.

Mager, R. F. (1992). Lernziele und Unterricht. *Preparing instructional objectives* (Rev. 2. ed., reprinted). London: Kogan Page.

Mills, A. J. (2010). *Encyclopedia of Case Study Research* (Bd. 1). Thousand Oaks: Sage Publications.

Mullis, I. V. S., Martin, M. O., Foy, P. & Arora, A. (2012). *TIMSS 2011 international results in mathematics*. Chestnut Hill, Mass.: TIMSS & PIRLS Internat. Study Center.

OECD (2013). *Was Schülerinnen und Schüler wissen und können. Schülerleistungen in Lesekompetenz, Mathematik und Naturwissenschaften* (PISA 2012 Ergebnisse, / OECD, Programme for International Student Assessment ; Bd. 1). Bielefeld: Bertelsmann.

Ott, B. (2000). *Grundlagen des beruflichen Lernens und Lehrens. Ganzheitliches Lernen in der beruflichen Bildung* (2. Aufl.). Berlin: Cornelsen.

Perez Martinez, J., Garcia Martin, J. & Sierra Alonso, A. (2014). Teamwork competence and academic motivation in computer science engineering studies. In IEEE (Hrsg.), *Global Engineering Education Conference (EDUCON)* (S. 778–783). IEEE.

Peters, L. & Moreno, A. M. (2015). Educating Software Engineering Managers - Revisited What Software Project Managers Need to Know Today. In *37th International Conference on Software Engineering (ICSE)*

Pfleeger, S. L. & Atlee, J. M. (2006). *Software engineering. Theory and practice* (3. Aufl.). Upper Saddle River, N.J: Pearson/Prentice Hall.

Plöger, W. (1999). *Allgemeine Didaktik und Fachdidaktik:* Uni TB.

Prince, M. J. (2004). Does Active Learning Work? A Review of the Research. *Journal of Engineering Education, 93*(3), 223–231.

Prince, M. J. & Felder, R. M. (2006). Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases. *Journal of Engineering Education, 95*(2), 123–138.

Puhlmann, H. (2008). *Grundsätze und Standards für die Informatik in der Schule. Bildungsstandards Informatik für die Sekundarstufe I ; Empfehlungen der Gesellschaft für Informatik e.V* (Log in, 150/151(2008),Beil). Berlin: LOG IN Verlag.

Rammstedt, B. & Ackermann, D. (Hrsg.) (2013). *Grundlegende Kompetenzen Erwachsener im internationalen Vergleich. Ergebnisse von PIAAC 2012*. Münster: Waxmann. Zugriff am 23.10.2015. Verfügbar unter http://nbn-resolving.de/urn:nbn:de:0168-ssoar-360687.

Rauner, F. (2011). *Messen beruflicher Kompetenzen* (Bildung und Arbeitswelt, Bd. 24). Münster: Lit.

Reich, K. (2012). *Konstruktivistische Didaktik. Lehr- und Studienbuch mit Methodenpool* (Pädagogik und Konstruktivismus, 5. Aufl.). Weinheim: Beltz.

Reischmann, J. (2003). *Weiterbildungs-Evaluation. Lernerfolge messbar machen* (Grundlagen der Weiterbildung Arbeitshilfen). Neuwied: Luchterhand.

Reiss, K. & Hammer, C. (2012). *Grundlagen der Mathematikdidaktik: Eine Einführung für den Unterricht in der Sekundarstufe:* Birkhauser.

Reißing, R. (2015). Softwarequalität. In J. Krahl & J. Löffl (Hrsg.), *Qualitäten* (Zwischen den Welten, Bd. 3, S. 87–108). Göttingen: Cuvillier Verlag.

Rindermann, H. (2003). Lehrevaluation an Hochschulen: Schlussfolgerungen aus Forschung und Anwendung für Hochschulunterricht und seine Evaluation. *Zeitschrift für Evaluation*. (2), 233–256.

Rogers, C. R. (1969). *Freedom to learn. A view of what education might become* (Studies of the person). Columbus, Ohio: C.E. Merrill Pub. Co.

Rupp, C. (2014). *Requirements-Engineering und -Management. Aus der Praxis von klassisch bis agil* (6. Aufl.). München: Hanser.

Saaty, T. L. (2001). *Fundamentals of decision making and prority theory with the analytic hierarchy process* (Analytic hierarchy process series, vol. 6, 2. Aufl.). Pittsburgh, PA: RWS Publications.

Savin-Baden, M. (2001). *Problem-based learning in higher education. Untold stories*. Buckingham: Society for Research into Higher Education & Open University Press.

Schmitt, U. (2013, 13. November). Obamacare entpuppt sich als gigantischer Flop. *Die Welt.* Zugriff am 12.03.2014. Verfügbar unter http://www.welt.de/politik/ausland/article121855210/Obamacare-entpuppt-sich-als-gigantischer-Flop.html.

Schmolitzky, A., Rick, D. & Forbrig, P. (Hrsg.) (2013). *HDI 2012 – Informatik für eine nachhaltige Zukunft. 5. Fachtagung zur Hochschuldidaktik der Informatik, 06. – 07. November 2012, Universität Hamburg* (Commentarii informaticae didacticae (CID)). Potsdam: Universitäts-Verlag Potsdam. Zugriff am 23.01.2014. Verfügbar unter http://publishup.uni-potsdam.de/opus4-ubp/files/6252/cid05.pdf.

Schneckenberg, D. (2008). *Educating Tomorrow's Knowledge Workers: The Concept of ECompetence and Its Application in International Higher Education:* Eburon.

Schwenn, K. (2005, 15. August). Prestigeprojekt mit Startschwierigkeiten. *Frankfurter Allgemeine Zeitung,* S. 6. Zugriff am 20.08.2015. Verfügbar unter http://www.faz.net/aktuell/wirtschaft/wirtschaftspolitik/verkehr-prestigeprojekt-mit-startschwierigkeiten-1250957.html.

Sedelmaier, Y. & Landes, D. (2012). A Research Agenda for Identifying and Developing Required Competencies in Software Engineering. In *15th International Conference on Interactive Collaborative Learning (ICL); 41st International Conference on Engineering Pedagogy;*

Sedelmaier, Y. & Landes, D. (2013). A Research Agenda for Identifying and Developing Required Competencies in Software Engineering. *International Journal of Engineering Pedagogy (iJEP), 3.*(2), 30–35.

Sedelmaier, Y. & Landes, D. (2014a). A Multi-Level Didactical Approach to Build up Competencies in Requirements Engineering. In B. Penzenstadler, S. Gregory & D. Landes (Hrsg.), *8th International Workshop on Requirements Engineering Education & Training (REET 2014), 22nd International Conference on Requirements Engineering (RE 2014)* (Bd. 1217, S. 26–34). CEUR Workshop Proceedings vol. 1217. Zugriff am 04.09.2014. Verfügbar unter http://ceur-ws.org/Vol-1217/paper4.pdf.

Sedelmaier, Y. & Landes, D. (2014b). A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies. SECAT - A Software Engineering Competency Assessment Tool. In IEEE (Hrsg.), *44th Frontiers in Education (FIE)* (S. 2065–2072)

Sedelmaier, Y. & Landes, D. (2014c). Practicing Soft Skills in Software Engineering. In L. Yu (Hrsg.), *Overcoming Challenges in Software Engineering Education* (S. 161–179). IGI Global.

Sedelmaier, Y. & Landes, D. (2014d). Software Engineering Body of Skills. In IEEE (Hrsg.), *Global Engineering Education Conference (EDUCON)* (S. 395–401). IEEE.

Sedelmaier, Y. & Landes, D. (2014e). Using Business Process Models to Foster Competencies in Requirements Engineering. In *27th International Conference on Software Engineering Education and Training (CSEE&T)* (S. 13–22)

Sedelmaier, Y. & Landes, D. (2015a). A Competence-Oriented Approach to Subject-Matter Didactics for Software Engineering. *International Journal of Engineering Pedagogy (iJEP), 5*(3), 34–44.

Sedelmaier, Y. & Landes, D. (2015b). Active and Inductive Learning in Software Engineering Education. In *37th International Conference on Software Engineering (ICSE)* (S. 418–427)

Sedelmaier, Y. & Landes, D. (2015c). Towards a Better Understanding of Learning Mechanisms in Information Systems Education. A Competence-Oriented Approach to Subject-Matter Didactics. In IEEE (Hrsg.), *Global Engineering Education Conference (EDUCON)* (S. 418–427). IEEE.

Sedelmaier, Y. & Landes, D. (2015d). Überfachliche Kompetenz im Software Engineering - Modellierung, Förderung und Messung in der Hochschulausbildung. In U. Riegel, S. Schubert, G. Siebert-Ott & K. Macha (Hrsg.), *Kompetenzmodellierung und -messung in den Fachdidaktiken* (S. 111–130). Münster: Waxmann.

Siebert, H. (1997). *Didaktisches Handeln in der Erwachsenenbildung. Didaktik aus konstruktivistischer Sicht* (2. Aufl.). Neuwied, Kriftel, Berlin: Luchterhand.

Siebert, H. (1999). *Pädagogischer Konstruktivismus. Eine Bilanz der Konstruktivismusdiskussion für die Bildungspraxis*. Neuwied, Kriftel: Luchterhand.

Siebert, H. (2006). Lernforschung - ein Rückblick. *Report - Zeitschrift für Weiterbildungsforschung*. (29), 9–14.

Sommerville, I. (2011). *Software Engineering* (9th ed). Boston: Pearson.

Steinweg, A. S. (2013). *Algebra in der Grundschule. Muster und Strukturen - Gleichungen - funktionale Beziehungen* (Mathematik Primarstufe und Sekundarstufe I + II). Berlin: Springer.

Strukturplan für das Bildungswesen. Verabschiedet auf der 27. Sitzung der Bildungskommission am 13. Febraur 1970 (1970) (Empfehlungen der Bildungskommission). Bonn: Bundesdruckerei.

ISO/IEC JTC 1/SC, 25010:2011 (2011, 01. März). Berlin: Beuth.

Terhart, E. (2002). Fremde Schwestern. *Zeitschrift für Pädagogische Psychologie, 16*(2), 77–86.

Terhart, E. (2009). *Didaktik. Eine Einführung*. Stuttgart: Reclam.

The Standish Group International. (1999). *CHAOS: A Recipe for Success,* The Standish Group International. Zugriff am 05.05.2014. Verfügbar unter https://www4.informatik.tu-muenchen.de/lehre/vorlesungen/vse/WS2004/1999_Standish_Chaos.pdf.

Vigenschow, U., Schneider, B. & Meyrose, I. (2011). *Soft Skills für Softwareentwickler. Fragetechniken, Konfliktmanagement, Kommunikationstypen und -modelle* (2. Aufl.). Heidelberg: dpunkt.verlag.

Weinert, F. E. (2001). Vergleichende Leistungsmessung in Schulen - eine umstrittene Selbstverständlichkeit. In F. E. Weinert (Hrsg.), *Leistungsmessungen in Schulen* (Beltz-Pädagogik, S. 17–31). Weinheim: Beltz.

Yourdon, E. & Constantine, L. L. (1979). *Structured design. Fundamentals of a discipline of computer program and systems design*. Englewood Cliffs, NJ: Prentice-Hall.

Anhang

Anhang

Anhang 1: Zentrale Publikationen

Anhang 2: Übersicht Copyright-Status

Anhang 3: Publikationsliste

Anhang 4: Erklärungen zu den Eigenanteilen

Anhang 5: Selbstständigkeitserklärung

Anhang 1: Zentrale Publikationen

Erläuterungen

Anhang 1.1

Sedelmaier, Y. & Landes, D. (2014). A Multi-Level Didactical Approach to Build up Competencies in Requirements Engineering. In B. Penzenstadler, S. Gregory & D. Landes (Hrsg.), *8th International Workshop on Requirements Engineering Education & Training (REET 2014), 22nd International Conference on Requirements Engineering (RE 2014)* (Bd. 1217, S. 26–34). CEUR Workshop Proceedings vol. 1217. Zugriff am 04.09.2014. Verfügbar unter http://ceur-ws.org/Vol-1217/paper4.pdf.

Anhang 1.2

Sedelmaier, Y. & Landes, D. (2014). A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies. SECAT - A Software Engineering Competency Assessment Tool. In IEEE (Hrsg.), *44th Frontiers in Education (FIE)* (S. 2065–2072)

Anhang 1.3

Sedelmaier, Y. & Landes, D. (2014). Using Business Process Models to Foster Competencies in Requirements Engineering. In *27th International Conference on Software Engineering Education and Training (CSEE&T)* (S. 13–22)

Anhang 1.4

Sedelmaier, Y. & Landes, D. (2015). Active and Inductive Learning in Software Engineering Education. In *37th International Conference on Software Engineering (ICSE)* (S. 418–427)

Anhang 1.5

Sedelmaier, Y. & Landes, D. (2015). Towards a Better Understanding of Learning Mechanisms in Information Systems Education. A Competence-Oriented Approach to Subject-Matter Didactics. In IEEE (Hrsg.), *Global Engineering Education Conference (EDUCON)* (S. 418–427). IEEE.

Anhang 1.6

Sedelmaier, Y. & Landes, D. (2015). Überfachliche Kompetenz im Software Engineering - Modellierung, Förderung und Messung in der Hochschulausbildung. In U. Riegel, S. Schubert, G. Siebert-Ott & K. Macha (Hrsg.), *Kompetenzmodellierung und -messung in den Fachdidaktiken* (S. 111–130). Münster: Waxmann.

Erläuterungen

Eigenanteil:

Der Anteil des eigenen Beitrags im Verhältnis zum Beitrag der anderen Ko-Autoren oder Ko-Autor-innen an der Erstellung des Artikels ist insgesamt unter angemessener Berücksichtigung aller Arten von Beiträgen nach folgender Klassifizierung anzugeben:

- Allein-Autorenschaft, wenn der eigene Anteil 100% beträgt.
- Überwiegender Anteil, wenn der eigene Anteil den Anteil jedes einzelnen anderen Ko-Autoren und jeder einzelnen anderen Ko-Autorin überwiegt und mindestens 35% beträgt.
- Gleicher Anteil, wenn (1) der eigene Anteil gleich hoch ist wie der von anderen Ko-Autoren oder Ko-Autorinnen, (2) kein weiterer Ko-Autor und keine weitere Ko-Autorin einen Anteil hat, der größer ist als der eigene, und (3) der eigene Anteil mindestens 25% beträgt.
- Wichtiger Anteil, wenn der eigene Anteil mindestens 20% beträgt, aber nicht Allein-Autorenschaft, überwiegenden Anteil oder gleichen Anteil darstellt.
- Geringer Anteil, wenn der eigene Anteil weniger als 20% beträgt.

Impact-Faktor:

Conferences are assigned to one of the following categories:

A* ─ flagship conference, a leading venue in a discipline area

A ─ excellent conference, and highly respected in a discipline area

B ─ good conference, and well regarded in a discipline area

C ─ other ranked conference venues that meet minimum standards

Australasian - A conference for which the audience is primarily Australians and New Zealanders

Unranked ─ A conference for which no ranking decision has been made

Anhang 1.1

Sedelmaier, Y. & Landes, D. (2014). A Multi-Level Didactical Approach to Build up Competencies in Requirements Engineering. In B. Penzenstadler, S. Gregory & D. Landes (Hrsg.), 8th International Workshop on Requirements Engineering Education & Training (REET 2014), 22nd International Conference on Requirements Engineering (RE 2014) (Bd. 1217, S. 26–34). CEUR Workshop Proceedings vol. 1217. Zugriff am 04.09.2014. Verfügbar unter http://ceur-ws.org/Vol-1217/paper4.pdf.

Impact-Factor (falls vorhanden): A

Eigenanteil:

Hauptautorin;

Konzeption des beschriebenen Ansatzes:	Gleicher Anteil (ca. 50 %)
Theoretische (pädagogische) Begründung des Ansatzes:	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung:	Gleicher Anteil (ca. 50 %)

A Multi-Level Didactical Approach to Build up Competencies in Requirements Engineering

Yvonne Sedelmaier, Dieter Landes
Faculty of Electrical Engineering and Informatics
University of Applied Sciences and Arts
96450 Coburg, Germany
{ yvonne.sedelmaier, dieter.landes }@hs-coburg.de

***Abstract*— Requirements engineering education at universities is a fairly difficult issue for various reasons. Among the most prominent causes is a lack of authenticity, i.e. too artificial settings that do not adequately mirror the complexity of real-world situations. We present an approach to requirements engineering education that tries to avoid some of these shortcomings, in particular by including requirements elicitation with real customers into an integrated didactic step-by-step approach. As it turns out, requirements engineering education is far more than assembling technical knowledge, but rather involves many non-technical skills that obtain a specific context-sensitive flavor in requirements engineering. Our didactic approach also addresses these skills, while resting on a sound pedagogical underpinning. Indications for the success of our approach are visible, e.g., in self-evaluations of the participants which are also summarized in the paper.**

***Index Terms*—requirements engineering, problem awareness, methodological skills, competencies, personal skills, real customers.**

I. Introduction

It is commonly accepted that requirements are top success factor in software engineering projects, but, conversely, also a major reason for project failures. Therefore, providing IT students with solid requirements engineering skills is of paramount importance. In practice, however, teaching and learning requirement engineering is not too easy. Part of the problem is the fact that good requirements are essential in complex real-life projects, but time and resource restrictions prohibit instructors from running many such projects – typically, there is only one such project during university education. Often, such a project comes late as a capstone project that ties together everything that should have been learned before. Unfortunately, learning requirements engineering only theoretically does not work well either. Students tend to view many important issues in requirements engineering as commonplaces and fail to see their importance.

It seems to be one of the big challenges for instructors to make requirements engineering education as descriptive as possible to make the matter more tangible for students. In particular, this encompasses mapping the complexity of real-world projects at least in part to a university context in such a way that the associated problems become evident for the intended audience.

In this contribution, we present the didactical approach that we developed for requirements engineering education at Coburg University of Applied Sciences. Core ingredients of our approach are a realistic and integrated setting, which includes writing a requirements document for a complex application and, as of late, eliciting requirements from real customers. In our specific setting, customers play a double role: in addition to simply providing requirements, they also act as external experts for communication issues. Another main characteristic of our approach is the extensive active involvement of students in the learning process. In particular the latter aspect has a solid theoretical underpinning in constructivist didactics. An additional characteristic of our approach is a strong emphasis on non-technical skills which are particularly relevant for requirements engineering, but also gain a very specific, context-sensitive shape in this particular domain.

The didactical approach that we conceived at Coburg seems to be quite successful since students value the importance of requirements engineering to a much higher extent and view themselves well equipped to deal with requirements engineering in practice. This finding is substantiated by a series of evaluations that we performed.

In the remainder of this paper, we will first analyze difficulties in teaching requirements in more detail before we characterize key features of our didactical approach and their pedagogical foundation. We discuss lessons learned both from the perspective of instructors and of students, before we give a short summary and an outlook to future work. Teaching Requirements Engineering

A. *Challenges in Teaching Requirements Engineering*

Although requirements engineering is a core ingredient of software engineering, it is fairly difficult to teach and to learn. Teaching and learning software engineering is generally restricted to small toy projects which mirror real world problems only to a limited extent. This is due to several reasons:

Early on, software engineering education often focuses on teaching and learning how to program a computer rather than

on requirements. Typically, programming assignments are small and clearly defined. Students design small pieces of software, often in small groups or even on their own. Assignments typically focus on a specific problem, e.g. specific element of the programming language such as arrays or loops, or particular algorithms. This means, software development assignments primarily focus on the technical aspects of programming and specific programming languages in a domain that is pretty familiar to students. As a consequence, requirements are supplied by the instructor, expressed clearly, and easy to understand since no unfamiliar terminology or unusual domain concepts are involved. This easily makes students believe that there is no need to bother with requirements since they generalize their programming experiences to real software development projects: There is no such thing like fuzzy requirements of stakeholders that use an unfamiliar terminology since students never came across such stakeholders. Consequently, there is a danger that students underrate the importance of requirements engineering since requirements engineering techniques do not solve any problem in their world. Often, students cannot even imagine problems that are rooted in insufficient requirements. And they do not believe instructors who report on their own practical experiences with what can go wrong with requirements. Students often think instructors exaggerate. Techniques they should learn in requirements engineering seem to be boring and useless since students mostly do not know why they need these techniques.

Furthermore, programming assignments tend to be isolated without relationship to other tasks. Even if there is more than one possible way to solve a problem the chosen approach will not have any consequences on following tasks. Students do not really need to balance reasons for or against alternative solutions. So, it would not matter if requirements were wrong or incomplete since students would not suffer from the consequences.

Even if, at a later stage, the focus shifts from programming to software engineering, and in particular to requirements engineering, the situation remains somewhat problematic: Due to time or capacity restrictions, the complexity of real world problems can hardly be reproduced in university education.

As one consequence, students usually do not perceive interdependences between requirements. They often suffer from the misconception that complexity scales up linearly. While a single use case is fairly easy to specify, dozens of use cases are not. If the number of requirements grows, so do the interdependencies between them.

In addition, students are in general given precise assignments and only need to apply known methods to solve a given problem. In university education students do not need to think about what the nature of the current problem actually might be. Students tend to take clear requirements for granted. For instance, it is quite easy for students to model a business process and extract use cases from it when the given example is very simple and clearly delimited. In "real" requirements engineering, requirements engineers first have to clarify the problem and then understand and solve it. They first have to elicit requirements before there is any point in thinking about technical solutions. Coming back to the example above, this means that there must be information on business processes in an organization before these processes can be modeled and taken as a basis for use cases. Frequently, this information is not readily available, but must rather be elicited from a range of appropriate stakeholders which frequently are not easy to identify. Students rarely face the problem of eliciting requirements from multiple groups of occasionally uncooperative stakeholders. Therefore, they do not see a problem in eliciting requirements.

All these challenges trace back to an insufficient match between scenarios in requirements engineering education and in real life. Given a restricted amount of time, it is quite difficult to expose students to examples which reflect real problems in requirements engineering. Requirements depend massively on the software that should be developed. Software engineering in university education mostly deals with developing a toy solution that will not be used in daily life. This also applies to requirements in university education: Requirements tend to be simple, and therefore requirements engineering seems to be unnecessary in students' opinions. Due to the lack of real customers students cannot imagine the complexity of and interrelationships between requirements within a large software engineering project.

B. *Didactical Approach in 2013*

At Coburg University, requirements engineering is a major issue in an elective course called "Software Modeling" which is offered in the second year of a bachelor program in informatics. Before enrolling into that course, students are required to take a compulsory introduction to software engineering in the preceding semester.

"Software Modeling" has been offered for several years and has been continuously evolving, including its didactical approach. The 2013 revision of the didactical approach aimed at improving students' understanding of requirements engineering and has been described in detail in [1].

For this approach we defined several intended learning outcomes in detail:

- Students shall acquire a more tangible impression of the term "requirements".
- Students shall understand the importance of requirements and shall be able to act accordingly.
- Students shall understand characteristic approaches to the specification of functional and non-functional requirements and their prioritization.
- Students shall understand the role of communication with other involved parties in requirements engineering.
- Students shall understand the role of business processes as a source of requirements.
- Students shall be able to collaboratively apply appropriate methods and notations in order so specify requirements for a sample software application.
- Students shall understand popular approaches to complexity and cost estimation for software systems.

Based upon the intended learning outcomes, the sequence of topics was restructured in order to focus on the given problem first before presenting solutions, relevant issues were illustrated on the basis of a continuous example, additional practical exercises were introduced, and predominantly passive learning settings shifted towards more active ones. The course emphasizes business process models as a source of requirements. Modeling processes puts software engineers in a position to extract requirements indirectly from an organizational workflow instead of, or in addition to, asking future users which functional and non-functional features the new software should exhibit.

This didactical approach includes the assignment to develop a requirements specification in teams of four or five students. To this end, students are exposed to a problem setting that they were sufficiently familiar with. For instance, the problem setting in 2013 was the derivation of requirements for a system to support the application, approval, and reimbursement for business trips. As a first step, students were required to develop business process models for the problem setting. The basic input for students consisted of an official leaflet which is used as a guideline for university members whenever they are about to go on a business trip. This brochure contains detailed rules for the application and reimbursement of a business trip and provides some details of how the process works. The teams of students extract distinct steps of the process before modeling them in a notation of their choice.

The business process models were subjected to a peer review. The process models of a peer group then served as a basis to extract use cases and fine-grained requirements.

Although this approach was successful from the perspectives of instructors as well as students, it still revealed some potential for improvement. Even though the assignment used a real world scenario, there is still no real customer from whom requirements must be elicited. And even though students obviously learned a lot in this course, not all teaching goals were completely achieved.

To sum up, in 2013 we applied several fundamental changes to our previous teaching approach in order to achieve our intended learning outcomes. Due to the fact that this new course design helped students to achieve these aforementioned goals, we decided to retain this didactical approach at large, and to refine it here and there.

So, in the 2014 iteration, we first refined our intended learning outcomes. So far, they were a little too abstract and we adapted the importance of some teaching goals again. While, for example, writing down given requirements is not the main focus any more, we now emphasize eliciting requirements form customers before writing them down.

C. *Intended Learning Outcomes*

The course "Software Modelling" aims at three main goals in addition to the existing intended learning outcomes:

- Students should understand the role and importance of requirements for their future careers. Students should develop problem awareness with respect to requirements engineering and recognize the importance of requirements and the difficulties in eliciting requirements. This teaching goal is assumed to be achieved if students are capable of eliciting requirements from future users, modeling business processes, and writing a requirements document.
- Students should enhance specific communication skills that are needed in requirements engineering. Students should be enabled to conduct a customer meeting in a goal-orientated way to elicit requirements. How can students elicit requirements which they did not "invent" themselves but are to be provided by a real customer? How can customers be prompted for information which may serve as a basis for requirements? How can requirements be documented and written down? How can students pass this challenge within a team (allocation of roles, etc.)?
- A third teaching goal is to strengthen self-reflection, self-organization, and self-responsibility of students. This is the basis for competence development [2].

As a consequence of the new prioritization of intended learning outcomes, a gap between them and the didactical design became evident so that some didactical fine adjustments became necessary.

II. Characteristics and Pedagogical Underpinning of a New Didactical Approach for Teaching Requirements Engineering

Based upon the experiences with the 2013 approach, we retained the structure of contents, activating learning elements, and a continuous example which culminates in writing a requirements document. Since active learning elements are commonly considered as a good approach for understanding abstract topics, we enhanced these aspects in the 2014 iteration. Students should play an active role in nearly every lesson instead of just listening to the instructor. During the lessons activation comes in by, e.g., small tasks that students need to deal with or by discussions between students and instructors.

One main weakness of the 2013 approach is the lack of eliciting requirements from a real future user or customer. So we refined our didactical setting mainly with respect to the following aspects.

A. *Eliciting Requirements from Real Customers*

One of the major drawbacks of our 2013 didactic approach was the fact that it did not address requirements elicitation. Several years back, we had tried to include this issue by having students elicit requirements from a peer team. Although this approach provided some insights with respect to difficulties of eliciting requirements, the whole setting was still artificial – students tended to be too cooperative in the role of a customer since they had no precise impression how real customers might act.

Therefore, we decided to bring in a real customer in 2014. Since we had chosen a system for managing offered training courses as application domain, we got in touch with a training provider in order to convince them to act as customers, a plan to which they happily agreed. We contacted a training and

consulting company with particular expertise in intra-project communication. This gave us an opportunity to include an additional aspect: Besides acting as a customer and reproducing typical behavioral patterns of customers in doing so, we had the chance to move to a meta-level right after the elicitation session. On this meta-level, the "customers", now in their role as communication experts, were to initiate a joint reflection with the student team on what had just happened in the elicitation session in terms of (un)successful communication.

In addition to being more realistic, students were expected to take the whole exercise more serious since they would not like to disgrace themselves in the face of externals. Furthermore, credibility was expected to increase since statements of external experts, based on their immediate practical experience, were deemed to have more weight than those of the instructor, who is latently alleged to exaggerate and, after more than ten years at university, to have lost immediate contact to what's happening in practice.

Students were split in two groups of approximately ten individuals and devoted a three-hour block for each team's elicitation session. About half of the session was planned for the actual elicitation of requirements from two customer representatives, and the other half, without the students knowing before, for an on-the-spot reflection of what went well and what did not. Students were asked to prepare for the elicitation meeting by pondering about good questions to ask, e.g. for identifying and clarifying business processes at the customers' site, and agree on an allocation of responsibilities and tasks within their team.

B. *Multi-level Teaching Approach*

When students enter this course, they already have some theoretical knowledge about specifying functional requirements through use cases [3].

We started the course with a first assignment that should be accomplished in teams of four students:

> *Exercise 1: Bidding for a software project*
>
> *A seminar provider intends to purchase a software system to manage his offered seminars. Imagine you as director of a software development company are asked to make an offer for such a software system.*
>
> *1. Think about your next four to five steps you would do, to prepare an offer. What would you do?*
>
> *2. How would you proceed? Give reasons why you decided for exactly this methods and approaches.*
>
> *3. Which problems might appear? What do you need to prepare that offer?*
>
> *Write down your results on a flipchart. (Working time: 30 minutes)*
>
> *Present your results in class.*

Students were supposed to take an active part in the course right from the start. This first exercise mainly aimed at raising awareness of requirements as an absolutely necessary prerequisite for bidding for a software project. Students should arrive at this insight by thinking about this exercise by themselves.

In a next step, students got an introduction to modeling business processes by using BPMN or event-driven process chains (EPCs).

Then students were split in two groups of, by and large, ten members each. Student teams were given a second assignment, namely they were supposed to elicit requirements from a real stakeholder, exchange their results, and build business process models on the information they received from the customer (see sec. III.A.). Process models were developed in a two-step approach: first, each team member developed an individual model before these individual models were merged and consolidated into a joint team model.

In the first exercise a lack of working techniques became evident. Therefore, we modified our second task by giving more precisely formulated briefings. For example, we added the following passage:

> *Exercise 2: Conduct a customer meeting*
>
> *[…]In preparation of the elicitation meeting with the customer, find an agreement on your intended course of action (among other things, your strategy to ask questions) and distribution of tasks. Clarify in the run-up the questions, you want to ask, the allocation of roles within your team, and the exchange of results at the end of the meeting. […]*

As an additional reaction to the two phases of the customer meeting (see sec. III.A.), which already included communication analyses on a meta-level, instructors decided to add a lecture session in order to further address communication and working techniques. In particular, this lesson put a focus on working techniques including allocation of roles and goal-orientation, approaches for preparing and conducting a customer meeting [4], question strategies, and communication techniques such as active listening [5]. This lesson was given in a pair-teaching format: the responsible instructor for this course with expertise in informatics acted jointly with an instructor with pedagogical background. As its main advantage, such a format offers the possibility to adapt and combine technical and non-technical knowledge and highlights inter-relationships between two disciplines to students. The customer meetings were analyzed again in a group discussion together with the students. Central questions were: "What went well? What would you do better next time?" Students realized by themselves that they should better prepare a meeting. Thus, they received information about structuring, preparing, and chairing a meeting. Furthermore, they learned about types of questions and question strategies to elicit needed information. This seems to be a good pedagogical approach because possible solutions are only presented after the need had actually arisen, i.e. students had already experienced a problem before they learned about possible solutions. Instead of teaching abstract and theoretical stockpiling knowledge, for which students typically do not know any use case, they could directly transform and apply the "newly acquired" knowledge.

As a preparation for the following session, students were also taught how to provide and to accept feedback, especially in a review process.

In parallel to the meta-analysis of the customer meeting, students got an assignment to model a business process in a notation of their choice. This task should be performed at home by each student individually. Following this, students should merge their individual business process models and derive a joint group model. The third exercise was to review their merged processes between teams of four or five students. To this end, they needed to remember and apply feedback rules. Without a-priori information about feedback and review processes students might feel accused and criticized.

Business process models are intended to serve as a source for requirements. Thus, students should now learn how to extract requirements from a business process model and write a requirements document. For this reason, a metaplan technique was used to activate students and collect contents of a requirements document as a first overview. Then several specific topics were worked out in class. During the following weeks, students were guided through several tasks which are necessary for writing a requirements document. Now that they know the context of single components they were gradually led to a complex document which contains all topics they learned before. Combining elements they develop over the time by themselves leads to a complete requirements document. Students had to work on individual and group exercises to repeat the learned contents in active work. Furthermore, they should apply theoretical learned knowledge and transform it into usable action knowledge.

In order to increase students' motivation, various exercises were associated with microcredits, i.e. a small bonus that may be used to improve the final grade in the exam.

C. *Pedagogical Underpinning*

There are indications that we learn

"- 10 % of what we read,
- 20 % of what we hear,
- 30 % of what we see,
- 50 % of what we hear and see,
- 70 % of what we say,
- 90 % of what we both say and do" [6] [7].

We choose this didactical multi-level approach to gradually build up students' competencies without overburdening them. Apparently, students are not used to structure their own working processes. With our approach we want to foster their self-organization and self-responsibility, and develop their communication skills and working techniques step by step. Students' previous knowledge and level of competence must be taken into account. This is a precondition to give students the possibility to further develop their competencies.

Designing an appropriate learning environment should be based upon constructivist principles. According to constructivist didactics, teachers act as coaches and can only give students room for their individual learning experience. Learning in this theory depends on the individual world and on the things a person learned before. Understanding arises from the interaction between the learner and the environment [8]. [8] conclude that "cognitive conflict or puzzlement is the stimulus for learning and determines the organization and nature of what is learned. […] Knowledge evolves through social negotiation and through the evaluation of the viability of individual understandings." It is necessary that the learner ties up his already existing knowledge and expertise to further develop it in his own way. Therefore, each student learns individual things according to his previous understandings, skills, and knowledge even if they experience the same learning situation.

Successful learning happens in learning situations which are adapted to students' previous skills and knowledge. Therefore, one of the main challenges in constructivist didactics lies in recognizing students' prior knowledge, then create appropriate learning environments, and adapt them specifically to the prior knowledge of students. In our didactical approach we gradually build up students' competencies by leading them through consecutive exercises, and strengthen their analytical skills as well as their self-organization by activating self-reflection processes.

Learning takes place when students consider the topics as relevant for their purposes [2]. As a consequence, they are interested in the issues and motivation for learning arises. Instead of teaching solutions for problems which students cannot even imagine, we make them see and understand the problems right at the beginning. After recognizing the problem they learn possible solutions to solve it and apply their new knowledge (learning by doing). In educational psychology these principles are main factors for successful learning [2].

III. Evaluation And Lessons Learned

A. *Instructors' Perspectives on Lessons Learned*

As instructors, we were surprised by the lack of students' work techniques we observed. Initially, we assumed that students had already exercised basic work techniques or basic communication skills at school. Yet, apparently this was not the case: The first given task turned out to be too complicated. This became evident during group work. While it was no problem for students to assemble in a group, they seemed to have severe difficulties to organize themselves within the group. Instructors expected that there would be a team leader, one student who writes down the results, one who presents them, one student as time keeper, etc. But teams started to work without structuring themselves, let alone assign roles to individuals. Even though instructors, at least from their perspective, provided a precisely formulated work assignment, results were fairly unstructured. Students read the assignment once at the beginning of the lesson, and then started to work without having a second look on the assignment. As a result, the results did not accurately fit the assignment. Teams should write down their final results on a flipchart and present them in class. Although students were advised to better use two sheets of a flipchart, some of them used both sides of a single sheet.

A severe lack of work techniques became also visible in the requirements elicitation session: students neither succeeded in allocating roles and tasks within their team, nor did they agree on question strategies before the meeting even though they

were provided with some advice what they were supposed to do. The provided hints were already a reaction to the perceived shortcomings in the first group assignments. As a consequence, we made our second exercise more precise and tried to give students more advice on what they could do to master the challenge. However, it was not enough and obviously did not help students at all. They were not able to prepare themselves for the meeting with the customers as we expected. Even though nearly all students were interested in participating in a customer meeting, some of them appeared completely unprepared. They neither had thought about possible questions they could ask, nor had they decided about team roles etc. All in all, these and several other observations led us to the assumption of lacking working techniques.

As a consequence, we added a teaching goal during the term that students should improve personal working techniques such as time management, endurance, self-organization, and structured course of action. Apparently, instructors' original intention to concentrate on fostering context-sensitive non-technical competencies in requirements engineering was too ambitious since prerequisites were missing.

Obviously, providing theoretical information about writing on a flipchart or organizing a team has no effect on students' learning processes. Rather, they need to experience some situations by themselves before learning becomes possible, including the possibility to make mistakes and learn from them. Apparently, students must reverse a flipchart sheet during the presentation of their work to recognize room for improvement. There is no point in telling them solutions before they experience the problem.

As instructors, we draw the conclusion to supply even clearer task assignments with very precise descriptions of what to do in future courses. Exercises should even be fairly fine-grained including precisely formulated steps what to do next.

Therefore, according to constructivist didactics, the third task was not simply "Write a requirements document". Instead of giving students a complex problem in one big chunk, we took our students by the hand and guided them through the process. The large task "requirements document" was partitioned into several smaller exercises, such as "Develop a use-case diagram", "Specify use cases", or "Derive functional requirements". Each week students got a new small task.

Adapting microdidactical elements during the lesson is based on the didactical principle of participant orientation ("Teilnehmerorientierung") [9] which is perfectly in line with constructivist didactics. [10] describes it as "reading" and „flexing". Reading means attentively observing students, while flexing concerns reacting on recognized requirements and needs. This generates an iterative process of adapting teaching and learning.

In constructivist didactical theory, teachers act as coaches for students and foster technical skills in combination with non-technical skills. Therefore, in future courses problem statements must be considered in more depth. It is necessary to work them out in more detail, and the nature of tasks in assignments needs to be well thought-out. Due to the fact that university cannot change students' previous knowledge and skills they bring into their studies, university teachers have to change their view on students' competencies and their learning processes.

Moreover, students often do not have any idea which methods and tools may help to elicit requirements from stakeholders. They do not know basic techniques to conduct a conversation which gives them needed information about processes in companies and the resulting requirements. During this course, instructors recognized that even if students knew in principle how to cope with the tasks, they looked helpless on it and had no idea what to do. Therefore, in addition to specifying assignments, instructors added a lesson to follow up on the customer meetings. Topics of this lesson were - in addition to methodological aspects - communication skills, such as questioning, and self-organization, such as preparing a meeting. As described above, pair teaching was chosen as didactical approach. In this lesson, students should get more action knowledge how they could master the given challenges. Course evaluation shows that students found this follow-up helpful. Several students appreciated this particular lesson when they were asked for things they considered necessary and important in an evaluation.

B. *Student Evaluation*

An intermediary evaluation of the course was conducted using the Software Engineering Competence Assessment Tool (SECAT) which was developed to evaluate students' competencies from multiple perspectives such as teachers, lecturers, or other students [11]. In this case, we used a self-estimation of students' competencies. SECAT also allows focusing on the assessment of one or more of nine criteria which are allocated to three levels of competence (see fig.1).

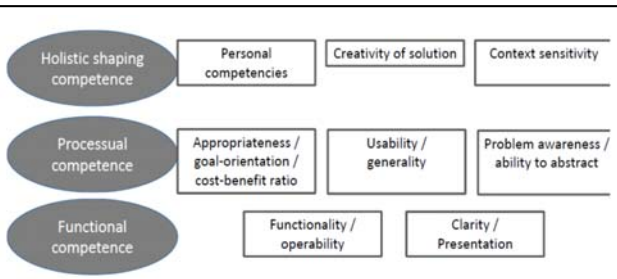


Fig. 1. Levels of competence in SECAT

Non-technical skills are high-level competencies in contrast to functional technical knowledge or the ability to present some content. In this case, we focus on problem awareness, context-sensitivity, and personal skills. Our teaching goals, namely improving problem awareness with respect to the importance of requirements engineering, fostering communication skills in context of customer meetings, and strengthening self-reflection as a basis of competence development are reflected in these three criteria. Each criterion was evaluated by means of 4 to 10

questions, according to the importance of the teaching goal (see tab. 1).

TABLE I. NUMBER OF SECAT QUESTIONS ACCORDING TO IMPORTANCE OF THE TEACHING GOAL

Criterion	Number of Questions
Problem awareness	4
Context sensitivity	10
Personal competencies	8
Creativity	2
Total	***24***

Each criterion is adapted to the specific situation and weighted by the number of questions per competence within the criterion. In this case, the main focus lies on context sensitivity which depicts in the competence "conducting a customer meeting". Table 2 shows the competencies which describe each criterion.

TABLE II. COMPETENCIES PER CRITERION

Criterion	Competencies
Problem awareness	Problem awareness / ability to abstract
Context sensitivity	• Moderation / Presentation • Conducting a customer meeting • Integrating in a team • Empathy • Endurance
Personal competencies	• Working techniques • Self-organization • Role allocation • Time management • Personal engagement • Goal orientation • Self-reflection
Creativity	• Creativity / Variety of methods

82 percent of a total of 20 students took part in the customer meeting, 88 percent modelled a business process on their own, and also 88 percent took part in the review process.

82 percent of our students find requirements engineering more interesting in comparison to the beginning of the term (see fig. 2). In the following figures, the left end of the scale means "Completely disagree", the right end means "Completely agree".

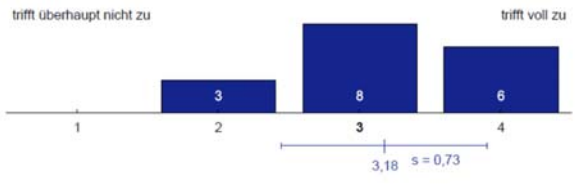


Fig. 2. Due to the course, I now view requirements elicitation more interesting than before the course

In last year's evaluation only 66 percent of our students agreed.

During the course, most of our students recognized the importance of requirements engineering for their future work (see fig. 3).

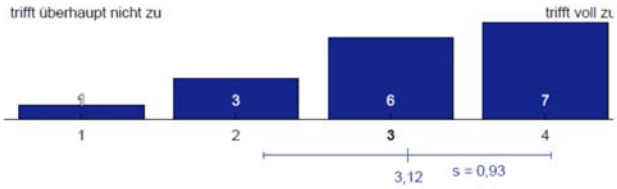


Fig. 3. Due to the course, I can now better appreciate the relevance of requirements for my work

As a result of the course nearly all students feel able to conduct a customer meeting for eliciting requirements (see fig. 4).

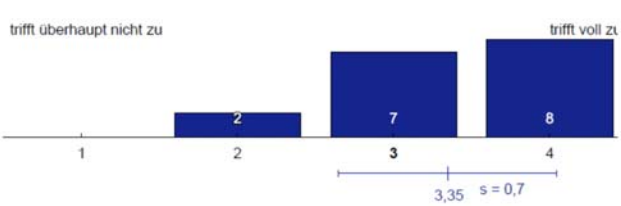


Fig. 4. Due to the course, I can now better conduct customer meetings.

Due to the course, students feel now able to reflect on situations and analyze them (see fig. 5).

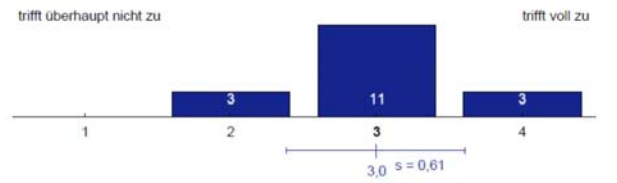


Fig. 5. Due to the course, I am now better equipped to analyze and understand specific conversations.

As a result of students' self-estimation with SECAT, competencies in the three main criteria, namely problem awareness with respect to the importance of requirements engineering, communication skills in customer meetings, and self-reflection, increased significantly (see fig. 6).

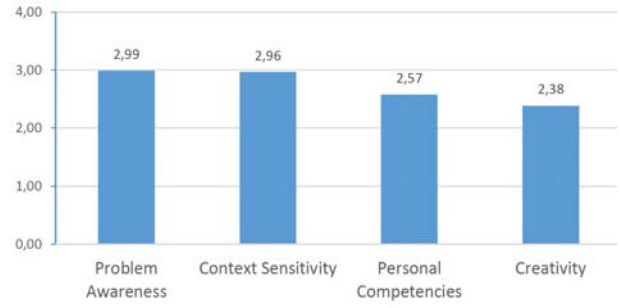


Fig. 6. Average increase of competence criteria

All in all, the evaluation showed a particular increase of competencies related to addressed intended learning outcomes (see fig.6 and fig. 7).

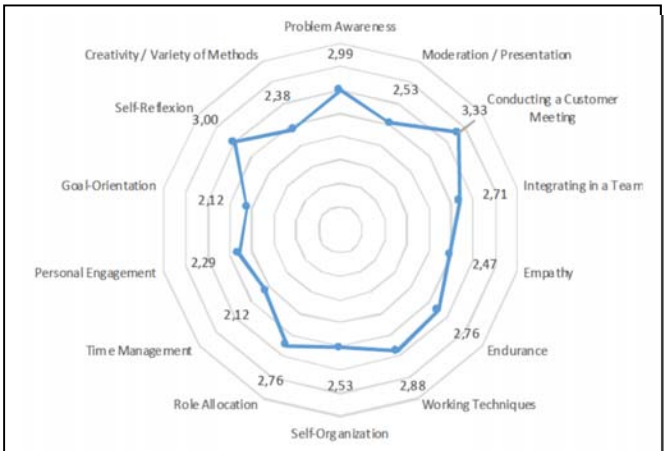


Fig. 7. Average increase of competencies over all students

Fig. 7 shows the largest increase of competence in "conducting a customer meeting", followed by "self-reflection" and "problem awareness". All values for these criteria are on a fairly high level of approximately 3 points.

Three out of 17 students (number 3, 5, and 10) did not take part in the customer meeting. Student nr. 10 with value 2.00 neither took part in the customer meeting, nor in the review of process models.

Evaluation results suggest that the chosen teaching approach allocated at constructivist didactics with consideration of psychological learning principles works well. Evaluation results indicate that the approach fosters students' competencies as explained in sec. II.C. Even intended learning outcomes which were added during the semester, such as working techniques, methodological skills, personal engagement, role allocation, or goal orientation, benefitted significantly. All students improved their competencies according to their self-estimation with values of at least 2 (see fig. 8).

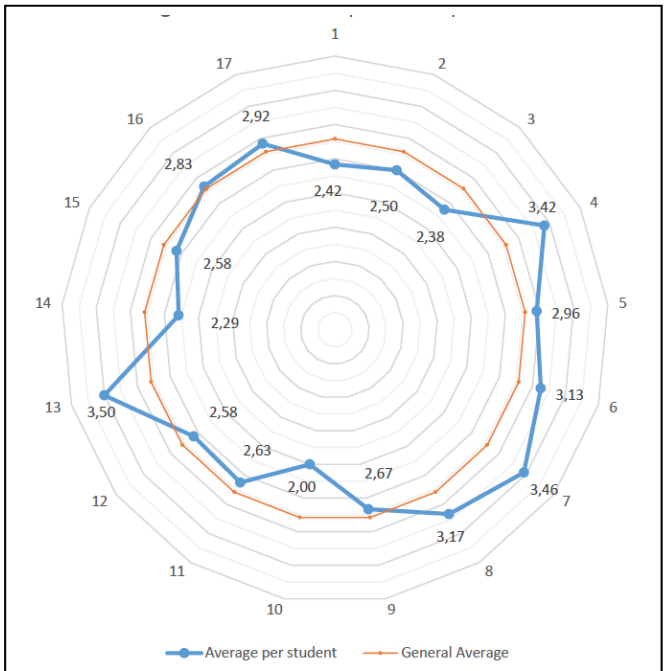


Fig. 8. Average increase of competencies per student

Also, even after being reluctant to be exposed to activating forms of learning, students seem to appreciate this format. In addition to statements in the evaluation which support this claim, this hypothesis is further substantiated by other indicators: 20 out of the 22 students who initially enrolled in the course actively participated in the course continuously, only 2 dropped out early. In addition, we had a regular physical attendance of 17 to 18 students in class throughout the complete semester, which is an unusually high rate. Since the course is an elective one without compulsory attendance, students would certainly have been scared away if they had not seen a real benefit in getting actively involved in the teaching and learning activities that we devised for the course.

IV. Summary and Outlook

We developed a didactical approach for requirements engineering education. Core ingredients of our approach are a realistic and integrated setting, which includes writing a requirements document for a complex application and, as of late, eliciting requirements from real customers. In our specific setting, customers play a double role: in addition to simply providing requirements, they also act as external experts for communication issues. Another main characteristic of our approach is the extensive active involvement of students in the learning process. In particular the latter aspect has a solid theoretical underpinning in constructivist didactics. An additional characteristic of our approach is a strong emphasis on non-technical skills which are particularly relevant for

requirements engineering, but also gain a very specific, context-sensitive shape in this particular domain.

Self-evaluations of participating students indicate significant increases in competencies that are relevant for requirements engineering and that we particularly targeted in the course. Currently, a final self-evaluation of students based at the end of the course is under way. In addition, we are just about to supplement and contrast the perspective of students with a SECAT-based evaluation from the instructor's perspective. Since the written examination associated with the course will be held shortly, we shall be in a position to correlate evaluation and examination results.

Although evaluation results so far indicate that the approach worked well, we still found potential for further enhancing our didactical approach.

It would be desirable to keep the meeting with a real customer on a regular basis for future courses. This seems to be the best way to make students understand the impact of requirements engineering. Unfortunately, organizational and financial difficulties have to be tackled before future students may be offered the opportunity for a real customer meeting. In a similar vein, it would be helpful if customers were not only available for an elicitation session, but also for, e.g., a review of business process models or requirements documents since this might uncover additional communication problems and expose potential for further competence development.

In future iterations of the course personal competencies such as working techniques and methodological skills should be taken into consideration right from the start. Instructors gained new insights into the level of basic skills of students. On this basis, they should adapt the didactical design to these additional intended learning outcomes, following the line of participant-orientation (see sec. IV). Our experiences indicate that university education must begin to foster basic skills at a much earlier point of time in bachelor programs.

Furthermore, it would be interesting to collect data from several cohorts of students. This would allow testing the hypothesis that this approach works well for similar groups of students.

ACKNOWLEDGMENT

We thank Ewa Sadowicz and Rainer Alt of EinfachStimmig, Nuremberg, for their active support.

The research project EVELIN is funded by the German Ministry of Education and Research (Bundesministerium für Bildung und Forschung) under grant no. 01PL12022A.

REFERENCES

[1] Y. Sedelmaier and D. Landes, "Using Business Process Models to Foster Competencies in Requirements Engineering," in Proc. 27th International Conference on Software Engineering Education and Training (CSEE&T), 2014, pp. 13–22.

[2] C. R. Rogers, *Freedom to learn:* A view of what education might become. Columbus, Ohio: Charles E. Merrill, 1969.

[3] A. Cockburn, *Writing effective use cases*. Boston: Addison-Wesley, 2001.

[4] J. W. Satzinger, R. B. Jackson, and S. D. Burd, *Introduction to systems analysis and design:* An agile, iterative approach, 6th ed. Mason, Ohio: Course Technology, 2012.

[5] U. Vigenschow, B. Schneider, and I. Meyrose, *Soft Skills für Softwareentwickler:* Fragetechniken, Konfliktmanagement, Kommunikationstypen und -modelle, 2nd ed. Heidelberg: dpunkt-Verlag, 2011.

[6] N. Green and K. Green, *Kooperatives Lernen im Klassenraum und im Kollegium:* Das Trainingsbuch, 3rd ed. Seelze-Velber: Kallmeyer, 2007.

[7] W. Niggemann, *Praxis der Erwachsenenbildung*. Freiburg: Herder, 1975.

[8] J. R. Savery and T. M. Duffy, "Problem Based Learning: An Instructional Model and Its Constructivist Framework," in *Constructivist learning environments: case studies in instructional design*, B. G. Wilson, Ed. 2nd ed, Englewood Cliffs N.J: Educational Technology Publications, 1998, pp. 135–148.

[9] U. Holm, *Teilnehmerorientierung als didaktisches Prinzip der Erwachsenenbildung - aktuelle Bedeutungsfacetten.* Available: http://www.die-bonn.de/doks/2012-teilnehmerorientierung-01.pdf (2014, May. 31).

[10] D. E. Hunt, "Lehreranpassung: 'Reading' und 'Flexing'," in Berichte, Materialien, Planungshilfen / Pädagogische Arbeitsstelle, Deutscher Volkshochschul-Verband, *Sensibilisierung für Lehrverhalten: Reaktionen auf D.E. Hunts „Teachers' adaption - 'reading' and 'flexing' to students"*, A. Claude, Ed, Frankfurt (Main): Pädag. Arbeitsstelle, Dt. Volkshochschul-Verb, 1986, pp. 9–18.

[11] Y. Sedelmaier and D. Landes, *A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies*. In Proc. 44th Frontiers in Education Conference (FIE 2014), Madrid, Spain, to appear.

Anhang 1.2

Sedelmaier, Y. & Landes, D. (2014). A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies. SECAT - A Software Engineering Competency Assessment Tool. In IEEE (Hrsg.), 44th Frontiers in Education (FIE) (S. 2065–2072)

Impact-Factor (falls vorhanden): B

Eigenanteil:

Hauptautorin;

Konzeption des beschriebenen Ansatzes:	Überwiegender Anteil (90%)
Theoretische (pädagogische) Begründung des Ansatzes:	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung:	Überwiegender Anteil (80%)

A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies

SECAT – A Software Engineering Competency Assessment Tool

Yvonne Sedelmaier, Dieter Landes
Faculty of Electrical Engineering and Informatics
University of Applied Sciences and Arts
96450 Coburg, Germany
{ yvonne.sedelmaier, dieter.landes }@hs-coburg.de

***Abstract*—Education invariably aims at developing competencies, technical as well as non-technical ones. As a consequence, there is also a need for methods that can be used to assess the quality of education faithfully. One possible approach is an assessment of whether intended learning outcomes are achieved, i.e. an investigation if the target audience possesses the desired competencies. Assessment of competencies, however, is tricky since competencies are often only vaguely defined. This paper presents SECAT, an approach to assess competencies, and particularly those needed for proper software engineering. To that end, SECAT builds on Rauner's approach for competency assessment in vocational education. Rauner's approach uses nine competency criteria, which are further refined by suitable issues that indicate to which extent a competency is, or should be, present.**

The main contribution of this paper lies in the adaptation and enhancement of this framework in order to make it useable in software engineering education. Adaptation and enhancements encompass issues such as team and individual assessments, integration of multiple perspectives from various groups of stakeholders, and product- and process-orientation. The paper also presents first insights from using SECAT in a pilot university course in software engineering.

Keywords—software engineering; software engineering education; assessment and evaluation approaches; quality of teaching and learning; didactical approaches; evaluation of didactical approaches; competence-orientation; SECAT;

I. INTRODUCTION

Right from the beginning, the Bologna Process triggered fundamental changes in higher education throughout Europe. Among other issues quality of teaching and learning at universities came into consideration. Evaluating quality of teaching and learning faces several challenges. First of all, criteria for good teaching have to be defined [1]. To do so, several approaches do exist: Some take teaching for good if various media such as e-learning approaches are employed. Others focus on integrating heterogeneous groups of students while still others emphasize the educational substance of a subject [2]. Some concentrate on activating students to participate in the learning process while others believe good teaching needs to entertain students [3]. Some instructors place their main emphasis on fostering students' competencies throughout a course, others primarily measure students' expertise in exams at the end of a course in order to obtain indicators for the quality of teaching and learning. Some take the process of learning as main issue and others emphasize the results of learning. Depending on the definition of good teaching, multiple approaches for evaluating teaching and learning, such as accreditation, student evaluation, and graduate surveys [4], are discussed and used in practice [5]. Each approach tries to provide evidence of good education. As a basis, we consider teaching and learning processes as "good" if students achieve our intended learning outcomes (ILO).

ILOs for software engineering are quite difficult to clarify because software engineering is a very complex subject with interrelationships to various other disciplines such as psychology. Software engineers require lots of technical and non-technical skills to cope with their daily work [6]. In addition to fundamental technical expertise and methodological knowledge, software engineers need to possess lots of personal and inter-personal skills, such as the ability to work in a team, individual working techniques, and the ability to structure abstract and complex issues before deciding on a possible solution [7].

As a consequence, intended learning outcomes in software engineering are very complex and involve various competencies which need to be transformed into action knowledge. Fostering these software-engineering-specific competencies is a challenge for higher education due to several reasons: In addition to the fact that various social and personal competencies are needed, students' lack of working experience causes a lack of problem awareness and problem-solving skills. Furthermore, the complexity of software engineering can hardly be mapped to a university context with limited resources. As a consequence, we started EVELIN (Experimental Improvement of Learning Software Engineering), a research project with the primary research goal to further develop and improve software engineering education at universities of applied sciences. Consequently, EVELIN can be allocated to empirical educational research and planning (Empirische Bildungsforschung) and focusses on teaching and learning research (Lehr-

978-1-4799-3922-0/14/$31.00 ©2014 IEEE

Lernforschung) to improve the quality of software engineering education at universities. It paves the way to a subject didactics (Fachdidaktik) for Software Engineering in higher education. The research design rests on two main pillars [8] which are repeated iteratively:

First, guided interviews are conducted to develop a competency profile for software engineers on the basis of Grounded Theory [9]. This process results in deeply understanding required technical and non-technical skills including their interrelationships for software engineering [7].

Then, in a second step, required competencies are fostered by developing and enhancing suitable didactical approaches [10] [11] [12]. To improve software engineering education, didactical approaches to better achieve the intended learning outcomes must be developed, applied, and evaluated.

Yet, evaluation of didactical approaches is a major challenge which can be tackled in different ways. According to Ditton [13] we decided to take the intended learning outcomes and the specifics of software engineering as criteria for evaluating teaching and learning.

As explained above, we assume that didactical approaches are appropriate if students' competencies come close to the intended learning outcomes during their studies. We adhere to this conceptual framework for evaluating software engineering education. Clear enough, no cause-effect relationship can be deduced: a tendency to a better command of desired skills cannot be ascribed uniquely to the didactical approach that has been employed. Yet, the didactical approach apparently does not do any harm if the majority of students possesses the required competencies to a larger extent than before being exposed to the didactical approaches in question. After defining these success criteria for good software engineering education, an evaluation tool for measuring these criteria is needed.

In short, we develop SECAT as a tool to evaluate didactical approaches in software engineering education at universities on the basis of increases of students' competencies, considering the specific profile of required software engineering competencies in the context of higher education.

This paper presents SECAT as a tool to assess competence-oriented education. SECAT builds on our previous experiences and an assessment methodology from vocational education, yet tries to avoid methodological shortcomings. The remainder of the paper first discusses the state of the art in evaluation and assessment in general and presents an approach to assess competency in vocational education. While the latter forms an interesting baseline, several extensions and modifications are required for being able to apply it to software engineering education. Details on SECAT and some results of applying SECAT to a capstone software engineering project will be presented, before we summarize our findings and give an outlook on future work.

II. Related Work

A. Evaluation of Didactical Approaches

Evaluating didactical approaches is difficult [14] - especially in subject didactics - and often lacks clear cause-effect-relationships in pedagogy. Kirkpatrick [15] presents four possible starting points with different coverage for evaluating didactical approaches: reaction (a.k.a. rating), learning, behavior, and results. SECAT primarily focusses on learning, yet with prospect to behavior and results, i.e. SECAT may be adapted to evaluate behavior and results of software engineering education in working live.

We decided in favor of a formative rather than summative evaluation [16] [14]. SECAT is a comparative approach to the extent that students' competencies are compared to the intended competency profile. SECAT evaluates the outcome of educational processes instead of intrinsic aspects of education such as learning processes itself or educational goals. It collects primarily quantitative data in order to "measure" effects of education. SECAT supports testing hypotheses rather than building them. Furthermore, SECAT includes both self- and external evaluation [1] [14] [16] [17].

The importance of assessing competencies reaches beyond judging the performance of individual students. Reischmann [14] views the main goal of evaluation in assessing the quality of educational processes, not in ranking individual learners. But in aggregated format, assessments of competencies may be compared to the course's intended learning outcomes in order to find out whether students, by and large, achieved the goals that instructors expected them to achieve. If a large share of students did not live up to some learning objective, this might be an indication that a chosen didactical approach requires some rework or needs to be changed entirely. Thus, an assessment scheme establishes possibilities to improve software engineering education, which is the overall goal of EVELIN, the project that we are currently running.

Consequently, SECAT is based on assessments of students' competencies.

B. Assessment of Students' Competencies

Learning inevitably comes along with some form of evaluation and often aims at acquiring or improving competencies. Evaluation then implies finding some form of evidence that a particular competency is actually exhibited. Yet, assessing competencies is generally difficult since competencies are somewhat fuzzy and rarely precisely defined. For instance, the capability to act adequately depends massively on the context of acting. A businessperson is competent in a different way than a social worker. What exactly are, say, engineers doing if they are competent? Does this mean they do have broad technical expertise? Or do we expect them to cooperate with stakeholders in an appropriate manner? But what exactly is an appropriate manner? How is competence characterized? Asking ten different people what they understand by competence, ten different answers will be given. Thus, defining competence is quite difficult, but nevertheless a prerequisite to assess competencies.

Various general frameworks for categorizing competencies have been proposed, e.g. [18] [19] [20]. So far, there seems to be little consensus of what competencies really are, and even more so which ones are needed to what extent for particular tasks, let alone for software engineering. This is particularly true for non-technical skills. Software engineering requires

various non-technical competencies [7] in addition to technical knowledge.

In university education, assessment of competencies is traditionally accomplished through some form of examination, either in writing or orally. This works fairly well for technical knowledge and lower levels of competence (e.g. "remember" in Bloom's taxonomy or "explain" in our own taxonomy [6]). For non-technical competencies and higher levels of competence, however, traditional examinations fall short of achieving the intended goal. This is particularly true in, e.g., hands-on team projects in software engineering education.

Social sciences like psychology and pedagogy have developed innumerable theories, methods, and tools for assessing competencies. Some authors rely on self-assessment [21], others developed psychometric tests to measure individual attributes [22], while others apply qualitative research methods like observations or interviews.

Many research projects focus on one or two competencies, often in a defined context, and try to develop tools to assess these specific competencies in some isolation. For example, Reeff developed a framework for assessing a single competence, namely adults' problem-solving competence [23]. Special attention is paid to the requirements of large-scale assessments. Several frameworks can be used both in the context of national studies and in international comparative studies of competencies such as the "Programme for an International Assessment of Adults' Competencies" (PIAAC, http://www.oecd.org/site/piaac) initiated by the OECD.

Generally, evaluation is based on either assessing students' competencies by one single instructor [24] or on self-assessments such as BEvaKomp [21].

In this paper, we refer specifically to software engineering education at universities of applied sciences, giving rise to a specific competency profile and the corresponding teaching objectives. Consequently, we need to take all software engineering competencies into account without an option to narrow things down to one or two competencies.

C. *Requirements for Evaluating Software Engineering Education*

In a first step, we aim at evaluating a capstone project in software engineering education. We run such a capstone project in the final year of our bachelor degree program in informatics [11] [12]. To evaluate such a capstone project, we originally devised a simple assessment scheme using Microsoft Excel. Basically, this assessment involved several criteria such as technical soundness of individual results and the overall solution, proper adaptation of a process model, or presentation and team skills. While this scheme already paid attention to individual as well as team performance on the one hand, and technical as well as non-technical competencies on the other, it lacks a sound methodological foundation. To make things worse, this scheme is only loosely linked with the intended learning outcomes for the course or more general frameworks such as SWEBOK [25] or SWEBOS [7].

Therefore, a suitable assessment framework for software engineering education needs to be employed which meets the following requirements:

- As a start, we want to assess a capstone project at the end of our Bachelor program in informatics. The course design encompasses bachelor and master students in joint teams. Therefore different teaching objectives for both groups of students arise, and a general framework is needed which can be applied to both groups and helps us to assess the respective technical and non-technical skills as well.
- The framework shall allow us to consider multiple perspectives. For instance, master students in the capstone project should receive a 360°-feedback from, say, bachelor students which they manage as a team leader. Furthermore, customers are asked for their impression with respect to team results and team working aspects.
- The assessment framework should consider team aspects as well as individual competencies.
- In any course, tangible results do matter. Yet, in capstone projects, the process that is followed to achieve these results is equally important. Likewise, methods and their usage need to be considered to assess the degree of competency when running a project.
- The assessment framework should take into account teaching goals and learning objectives. SWEBOS [7] serves as a guideline for software engineering competencies. On the basis of the assessment framework, didactical approaches to learning software engineering should be improved and further developed in a competency-oriented manner [8]. In order to be able to do so, the framework should highlight deficiencies and shortcomings as candidate areas for further adjustment and improvement.
- The framework does not need to support the exact and valid measurement of individual competencies.

To sum up, SECAT allows to evaluate the improvement of both technical and non-technical skills. Moreover, SECAT considers team results as well as individual performance. Evaluation with SECAT takes multiple perspectives such as from students, lecturers, and customers, into account. SECAT takes students' competencies as a starting point for evaluating the effectiveness of didactical approaches in software engineering.

As it turned out, none of the existing approaches matches these requirements without being adapted.

III. RAUNER'S FRAMEWORK FOR ASSESSMENT OF OCCUPATIONAL COMPETENCE

A. *Characteristics of Rauner's Framework*

Rauner's approach [26] from vocational education might serve as a promising starting point for adaptation and enhancement. This approach rests firmly on educational theory and is validated empirically. It is based on a competency model which

is characterized by three dimensions, namely levels of competence, content dimension, and action dimension [27].

1) Levels of Competence

Heinemann and Rauner [27] define nine competency criteria and map them to three levels of competence, namely functional competence, processual competence, and holistic shaping competence (see fig. 1). There is also a match to the six Dreyfus levels which distinguish beginner, advanced beginner, competent, proficient, and expert as stages of competence development [28].

2) Content Dimension

This dimension of Rauner's competence model refers to the teaching and learning contents of a subject, in our case software engineering. "There is the 'objective' dimension of content. As is commonly known, a competence as a latent construct can only be seen via performance. This performance has to be displayed in the area of some given content - *what* kinds of problems a given competence can tackle, may it be general school subjects or occupational competence." [27] In other words, the contents establish the context in which the competency is shown.

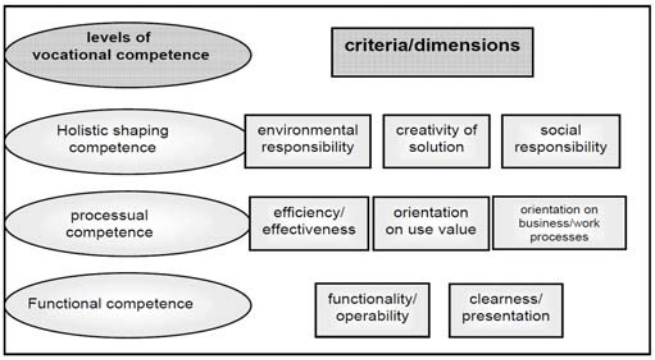


Fig. 1. Vocational Competence Model [27], p.143

3) Action Dimension

Tasks operationalize the content dimension und relate it to the level of competence. The action dimension "[…] is 'subjective' dimension on *how* a given problem is treated. Individuals act differently when encountering a given problem due to the competence they already acquired." [27]

Rauner creates open test tasks with a limited solution space to assess the actual level of competence of his test subjects. Typically, 5 items per criterion, rated on a 4-level scale, are applied to measure vocational competencies [24].

B. Evaluation of Rauner's Framework

Rauner's approach contains several promising ingredients, although some aspects still need to be adapted to the requirements of software engineering education:

- Criteria are helpful, but must be transformed from vocational education in a technical domain to software engineering.
- The framework considers technical expertise as well as non-technical skills and competencies.
- Three levels of competence are sufficient to cover the range from novices to experts in competence assessments.
- A sound pedagogical underpinning exists.

As a downside, some aspects deserve to be added:

- Even though inter-rater reliability is high in Rauner's approach, it lacks multiple perspectives. Not only instructors should assess competencies in software engineering, but rather peers or other stakeholders should complement their rating.
- Not only results, but also processes should be taken into account.
- The outcome of a task does not reflect each required competence appropriately, especially if team results are assessed as well.

IV. An Assessment Framework for Software Engineering Education: **S**oftware **E**ngineering **C**ompetencies **A**ssessment **T**ool (SECAT)

Assessment of competencies generally encompasses two different dimensions: on the one hand, the perspective on a course in its entirety including the effects of the didactical approach and, on the other hand, the personal competencies of individual students. The first dimension builds upon an aggregated competence evaluation of all students in a course. For example, students' competencies early in a course are compared to their competencies at the end of the course. Thus, conclusions on the didactical approach applied in this course can be drawn from a delta-analysis. Furthermore, different cohorts enrolling in the same course in different years yield indications on the effects of modified didactical elements in the course over time.

The second dimension aims at assessing individual, personal competencies of single students. These personal competencies may be expressed relative to other students' competencies in the same course or, for example, measured on an absolute scale for one student. Instructors benefit from other perspectives on students' competencies being revealed when grading their performance. Even though assessing individual students' competencies is not the primary focus of SECAT, a consensus among raters may substantiate the validity and reliability of SECAT.

SECAT adapts and extends the approach for measuring competencies that is described in sec III.A with respect to the requirements in software engineering education at Coburg University of Applied Sciences. Since a university degree is significantly different from vocational education, adaptations are inevitable. The existing approach requires several extensions and adaptations. First, criteria had to be adapted to software engineering (see table 1), before items for multi-perspective assessments are developed.

A. Adaption of criteria

Criteria in SECAT are based on our intended learning outcomes for software engineering which are detailed in [7].

TABLE I. TRANSFER OF CRITERIA FROM RAUNER'S VOCATIONAL COMPETENCE MODEL TO SECAT

Criterion in Rauner's Approach	Criterion in SECAT
Functional competence	***Functional competence***
Functionality / operability	Functionality / operability
Clearness / presentation	Clarity / presentation
Processual competence	***Processual competence***
Efficiency / effectiveness	Appropriateness / goal-orientation / cost-benefit ratio
Orientation on use value	Usability / generality
Orientation on business / work processes	Problem awareness / ability to abstract
Holistic shaping competence	***Holistic shaping competence***
Environmental responsibility	---
Creativity of solution	Creativity of solution
Social responsibility	Context sensitivity
---	Personal competencies

1) Transfer of functional competence to SECAT

The criteria "functionality / operability" and "clarity / presentation" characterize the level of functional competence. Technical knowledge becomes manifest in context-free, factual, and technical information. In essence, functional competence is concerned with knowing *that* something has to be accomplished.

2) Transfer of processual competence to SECAT

The processual level describes procedural technical expertise, i.e. knowledge *how* to do something. Work tasks are considered in the context of organizational workflows. Processual knowledge establishes vocational acting competence. Thus, Rauner's criteria for the processual level need to be adapted to software engineering. In a software engineering context, efficiency / effectiveness can be characterized by appropriateness, goal-orientation, and cost-benefit ratio. Students are capable of finding an appropriate solution, of planning their time realistically, and of setting goals for themselves (SWEBOS S) [7]. Furthermore, they can relate required effort to expected benefit.

Orientation on use value turns into usability and generality. Students are supposed to pay attention to the solution's fitness for the intended purpose. They also understand the importance of documentation in software engineering in order to adapt and enhance software systems.

In SECAT, Rauner's orientation on business / work processes turns into problem awareness and the ability to abstract. SWEBOS characterizes this competence as follows [7]: "These competencies aim at abstracting complex problem settings and other professional disciplines. Thus, [software engineers] need to be willing to and capable of thinking outside the box, and of understanding and accepting the significance and necessity of allegedly strange procedures and artifacts." Also, nonfunctional requirements should be taken into account when developing a software system.

3) Transfer of holistic shaping competence to SECAT

Holistic shaping competence denotes awareness of the complexity of work tasks (know *why* to do something) and the capability to decide on a solution for a problem after analysing various potential options. Holistic shaping competence also encompasses a reflective component.

Environmental responsibility has no equivalent in SECAT.

Rauner's creativity of solution is adopted in SECAT. Creativity of solution in SECAT is described in [7] as the capability to apply one's individual knowledge and skills to specific and novel situations where no cook-book recipes are applicable. This presumes the awareness of problems in order to apply methods and tools, and to develop new solutions.

Social responsibility [26] is transformed into SECAT's context sensitivity. This competence encompasses several interpersonal skills such as the competence for professional collaboration with others as described in [7], including aspects of teamwork and social interaction with stakeholders.

Some competencies are missing in Rauner's approach. SECAT needs to consider personal competencies as well since software engineering requires, among others, communicative competence and self-organization, i.e. competencies for structuring one's own way of working.

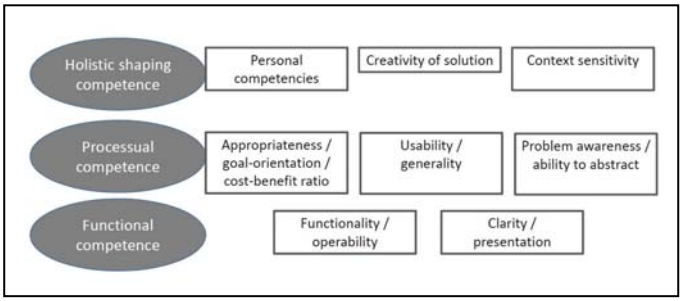


Fig. 2. Levels of Competence in SECAT, adapted from [27]

B. Development of assessment items

After adapting the competency model, items for rating competencies must be developed. These items should exhibit several characteristics. The assessment framework should not contain too many items per perspective because there are already nine criteria to be assessed. Assessing each criterion by asking more than five questions on average would be boring and time-consuming for assessors.

As a pilot, a capstone project was assessed. Meanwhile, SECAT has been applied to additional courses.

The assessment should indicate levels of competence for each criterion. Different perspectives, e.g. from customers or instructors, should complement each other and yield a comprehensive picture. As a consequence, items can be assigned to a single criterion, but are individually combined to mirror various perspectives on assessing competencies (customer, instructor, etc.) or different courses with different focus, ranging from beginners' to expert courses.

In contrast to Rauner's approach, we also take team aspects into account in addition to individual competence assessments.

Furthermore, items in our approach not only assess solutions of typical tasks, but also consider the process of how students arrived at a result.

In our pilot capstone project, bachelor and master students work together, yet with different teaching goals. Therefore, the assessment focuses on different competencies for bachelor and master students. Bachelor students' teaching goals emphasize technical expertise, while master students should improve their leadership skills. Criteria stay the same for both bachelor and master students, but weights and items used for assessing competence levels are different for the two groups.

In addition to instructors' perspectives, two additional points of view are used. In general, each perspective contains items assessing all nine criteria, however, the number of items fluctuates strongly. For example, customers cannot assess individual competencies or the creativity of the solution since they neither are close enough to the project internals, nor do they have sufficient technical expertise in informatics. At the end of the capstone project, customers primarily assessed the criteria goal-orientation, cost-benefit ratio, usability / generality, and context-sensitivity. Furthermore, we added a peer perspective. Bachelor students assessed their team leader (i.e. one of the master students), while master students assessed all of their team members. Bachelor students assessed all criteria except for usability / generality and creativity of solution due to lack of technical expertise.

For each of the nine criteria up to 11 questions according to the assessment perspective are formulated. On average, three questions per criterion were developed (see table 2).

TABLE II. NUMBER OF QUESTIONS PER PERSCPECTIVE AND CRITERION

Criterion	Number of Questions per perspective		
	Lecturer	*Customer*	*Student*
Functional competence			
Functionality / operability	4	2	4
Clarity / presentation	2	3	3
Processual competence			
Appropriateness / goal-orientation / cost-benefit ratio	4	6	11
Usability / generality	4	5	0
Problem awareness / ability to abstract	4	2	5
Holistic shaping competence			
Context sensitivity	6	6	3
Personal competencies	6	1	7
Creativity of solution	4	2	0
Total	***34***	***27***	***33***

Customers primarily assessed team aspects while students assessed individual competencies, e.g. of the master student who led their team. Instructors were asked questions on both team and individual aspects.

For example, lecturers were asked the following questions to assess personal competencies:

- The student was capable of reflecting her own behaviour and of drawing conclusions for her own personal development.
- The student was capable of handling (alleged) setbacks.
- The student was capable of structuring and organizing her own work well.
- The student was capable of motivating herself continuously and of setting goals for herself.
- The student was capable of making compromises for the benefit of a constructive overall solution.
- The student was capable of mastering her tasks in time and on a high quality level.

V. FIRST RESULTS/LESSONS LEARNED

Using this approach, we received assessments of each criterion from several different perspectives (see fig. 3.)

First results show that the assessments of competence criteria do not differ much across different perspectives, i.e. are externally consistent with each other (see fig. 3). Fig. 3 shows the multi-perspective competence assessment of a sample master student in our capstone project.

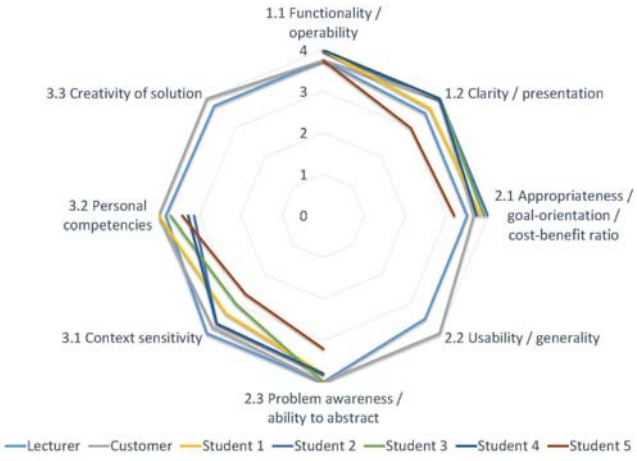


Fig. 3. Multi-perspective Assessment Results from SECAT

We wanted to discover possible mismatches between assessments of particular criteria from different perspectives. In addition, we strive for a broad view on competencies. To this end, we used SECAT to collect qualitative data and compare different views on single competence criteria or two different instructors' estimates. In our capstone project we were able to compare perspectives from lecturers, customers, and fellow students with respect to software engineering competencies of students. As we hoped these views do not differ a lot, which implies high inter-rater reliability.

Thus, SECAT allows us to conclude that in capstone projects students learn a lot. Project work seems to be a good didactical approach to foster the intended technical and non-technical competencies, in particular multiple high-level competencies in software engineering. These conclusions are substantiated by multiple consistent perspectives.

Non-technical skills cannot be measured exactly. Therefore, we use multi-perspective estimations which can be compared to each other in order to identify significant differences between assessors or provide an "objective" view on someone's competencies. SECAT may relativize single assessor's opinions about individual students' competencies by providing views from additional perspectives.

In our opinion, SECAT meets our requirements well since it is based on a sound competency model. Furthermore, it is customized to software engineering competencies and teaching goals.

Even though the effort is relatively high to adapt the approach from industrial-technical occupations to personal service occupations [29] or even university education in software engineering, the benefit justifies the pain of transferring this approach.

VI. Summary and Outlook

This paper presented SECAT, an approach to evaluate software engineering education by assessing students' competencies. SECAT builds upon an approach from vocational education that proved its value over many years. Due to its different domain, the latter approach, however, had to be adapted and extended in several ways in order to be usable for software engineering. In particular, SECAT considers team achievements as well as individual ones, integrates multiple perspectives from various groups of stakeholders, and pays attention to the outcome of a task as well as the process that was used to solve the task. Also, it is worth mentioning that SECAT, like Rauner's original approach, covers technical as well as non-technical competencies in an integrated framework.

Thus, SECAT can be used for assessing competencies of individuals, but is primarily a tool that may highlight shortcomings in didactical approaches in university education that might be eliminated by changing didactical elements in a competence-oriented manner.

From our point of view, the latter is the primary use of SECAT. In order to obtain indications for further improvement of software engineering education, it is sufficient to focus on nine criteria. It is neither necessary to develop a large-scale assessment for individual competencies, nor to define items which suit all perspectives, such as lecturers, customers, and peers.

In the near future, we will apply the framework for assessing competencies in additional software engineering courses and refine and enhance assessment questions.

As future work, we will also collect and analyze additional data to improve the validity of SECAT. First results suggest the hypothesis of high inter-rater reliability. Yet, this hypothesis needs to be tested. Furthermore, criteria need to be weighted appropriately – for simplicity, all criteria are currently assumed to be equally important. Weights should be determined according to, e.g., the assessment perspective, the teaching goals, or the study term. A student in the first term is expected to exercise or develop other competencies than a final year student. During the bachelor degree program the focus turns from learning technical knowledge in the first terms to gradually transferring it into work tasks and developing vocational action competencies. After adding the possibility to emphasize different criteria in the assessment of different students, SECAT can be applied in all study terms of software engineering education.

SECAT provides data for a "Competence Repository" (CORE) [30] which provides capabilities for educational data mining. Basically, SECAT assesses competencies, or increases in competencies for individual students. These data are collected with SECAT and provide the basis for, e.g., delta analysis of groups of students in CORE or trend analysis. In CORE competencies of single students are aggregated to a big picture and show competencies of groups of students. In a next step, we can compare, e.g., team results and individual competencies of single students. Are there any relationships or patterns? Do different cohorts develop similarly? We expect interesting findings from investigating these questions that will allow us to further improve software engineering education.

Acknowledgment

We want to thank Roland Mohl and Christian Zürl for their contribution to tool support for SECAT.

The research project EVELIN is funded by the German Ministry of Education and Research (Bundesministerium für Bildung und Forschung) under grant no. 01PL12022A.

References

[1] H. Rindermann, "Lehrevaluation an Hochschulen: Schlussfolgerungen aus Forschung und Anwendung für Hochschulunterricht und seine Evaluation," *zeitschrift für Evaluation*, no. 2, pp. 233–256, http://www.zfev.de/fruehereAusgabe/ausgabe2003-2/artikel/ZfEv2-2003_5-Rindermann.pdf, 2003.

[2] W. Klafki, "Didactic analysis as the core of preparation of instruction (Didaktische Analyse als Kern der Unterrichtsvorbereitung)," *Journal of Curriculum Studies*, vol. 27, no. 1, pp. 13–30, 1995.

[3] K. Kirchgeßner, *Gute Lehre:* Frischer Wind an deutschen Hochschulen, 1st ed. Bonn: HRK, 2011.

[4] C. Gold, Y. Sedelmaier, and J. Abke, "A Retrospective Course Survey of Graduates to Analyse Competencies in Software Engineering," in Global Engineering Education Conference (EDUCON): IEEE, 2014, pp. 100–106.

[5] H. Schraeder and W. Dreger, Eds, *Hochschule entwickeln, Qualität managen: Studierende als (Mittel)punkt:* Die Rolle der Studierenden im Prozess der Qualitätssicherung und -entwicklung. Bonn: HRK, 2005.

[6] Y. Sedelmaier, S. Claren, and D. Landes, "Welche Kompetenzen benötigt ein Software Ingenieur?," in Software Engineering im Unterricht der Hochschulen 2013, 2013, pp. 117–128.

[7] Y. Sedelmaier and D. Landes, "Software Engineering Body of Skills," in Global Engineering Education Conference (EDUCON): IEEE, 2014, pp. 395–401.

[8] Y. Sedelmaier and D. Landes, "A Research Agenda for Identifying and Developing Required Competencies in Software Engineering," *International Journal of Engineering Pedagogy (iJEP)*, vol. 3, no. 2, pp. 30–35, 2013.

[9] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine Transaction, 2009.

[10] Y. Sedelmaier and D. Landes, "Using Business Process Models to Foster Competencies in Requirements Engineering," in 27th International Conference on Software Engineering Education and Training (CSEE&T), 2014, pp. 13–22.

[11] Y. Sedelmaier and D. Landes, “Practicing Soft Skills in Software Engineering,” in *Overcoming Challenges in Software Engineering Education*, L. Yu, Ed.: IGI Global, 2014, pp. 161–179.

[12] D. Landes, Y. Sedelmaier, V. Pfeiffer, J. Mottok, and G. Hagel, “Learning and teaching software process models,” in Global Engineering Education Conference (EDUCON), 2012, pp. 1153–1160.

[13] H. Ditton, “Evaluation und Qualitätssicherung,” in *Handbuch Bildungsforschung*, R. Tippelt, Ed. 2nd ed, Wiesbaden: VS Verlag für Sozialwissenschaften, 2009, pp. 607–623.

[14] J. Reischmann, *Weiterbildungs-Evaluation:* Lernerfolge messbar machen. Neuwied: Luchterhand, 2003.

[15] D. L. Kirkpatrick and J. D. Kirkpatrick, *Evaluating training programs:* The four levels, 3rd ed. San Francisco, CA: Berrett-Koehler, 2006.

[16] C. Wulf, “Curriculumevaluation,” in Erziehung in Wissenschaft und Praxis, vol. 18, *Evaluation: Beschreibung und Bewertung von Unterricht, Curricula und Schulversuchen*, C. Wulf, Ed, München: R. Piper, 1972, pp. 15–37.

[17] L. J. Cronbach, “Evaluation zur Verbesserung von Curricula,” in Erziehung in Wissenschaft und Praxis, vol. 18, *Evaluation: Beschreibung und Bewertung von Unterricht, Curricula und Schulversuchen*, C. Wulf, Ed, München: R. Piper, 1972, pp. 41–59.

[18] B. S. Bloom, M. Engelhart, E. Furst, W. Hill, and D. R. Krathwohl, *Taxonomy of Educational Objectives – The Classification of Educational Goals – Handbook 1: Cognitive Domain*. London, WI: Longmans, Green & Co. Ltd, 1956.

[19] L. W. Anderson, D. R. Krathwohl, and B. S. Bloom, *A taxonomy for learning, teaching, and assessing:* A revision of Bloom's taxonomy of educational objectives. Princeton, N.J: Longman, 2000.

[20] J. Erpenbeck, Ed, *Handbuch Kompetenzmessung:* Erkennen, verstehen und bewerten von Kompetenzen in der betrieblichen, pädagogischen und psychologischen Praxis, 2nd ed. Stuttgart: Schäffer-Poeschel, 2007.

[21] E. Braun, *Das Berliner Evaluationsinstrument für selbsteingeschätzte studentische Kompetenzen (BEvaKomp)*. Göttingen: V & R Unipress, 2008.

[22] C. V. Gipps, *Beyond testing:* Towards a theory of educational assessment. London, Washington, D.C: Falmer Press, 1994.

[23] J.-P. Reeff, *The assessment of problem solving competencies*. Bonn: Deutsches Institut für Erwachsenenbildung (DIE), 2006.

[24] A. Maurer, F. Rauner, and L. Heinemann, “Ensuring Inter-Rater Reliability in a Large Scale Competence Measurement Project in China,” in Innovative apprenticeships*: Promoting successful school-to-work transitions*, Berlin: Lit, 2010, pp. 157–160.

[25] P. Bourque and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge Version 3.0 - SWEBOK*. Available: http://www.computer.org/ieeecs-swebokdelivery-portlet/swebok/SWEBOKv3.pdf?token=RkjKSKMu0hA2hyGQxx5lHHB1LTkirNPu (2014, Mar. 13).

[26] F. Rauner, *Messen beruflicher Kompetenzen*. Münster: Lit, 2011.

[27] L. Heinemann and F. Rauner, “Occupational Identity and Motivation of Apprentices in a System of Integrated Dual VET,” in Innovative Apprenticeships*: Promoting Successful School-to-Work Transitions*, Berlin: Lit, 2010, pp. 141–144.

[28] F. Rauner, “Practical Knowledge and Occupational Competence,” *european journal of vocational training*, no. 40, pp. 52–68, http://www.cedefop.europa.eu/EN/Files/40-en.pdf, 2007.

[29] F. Rauner, “Kompetenzentwicklung und -messung in beruflichen Bildungsgängen und Handlungsfeldern,” Offenbach, Nov. 2011.

[30] M. Koch and D. Landes, “Design and Implementation of a Competency Repository,” in *New perspectives in information systems and technologies, volume 1*, Heidelberg: Springer, 2014, pp. 249–255.

Anhang 1.3

Sedelmaier, Y. & Landes, D. (2014). Using Business Process Models to Foster Competencies in Requirements Engineering. In 27th International Conference on Software Engineering Education and Training (CSEE&T) (S. 13–22) © 2014 IEEE. Reprinted, with permission, from Sedelmaier, Y. & Landes, D., Using Business Process Models to Foster Competencies in Requirements Engineering, IEEE 27th Conference on Software Engineering Education and Training, 2014

Impact-Factor (falls vorhanden): C

Eigenanteil:

Hauptautorin;

Konzeption des beschriebenen Ansatzes:	Gleicher Anteil (ca. 50 %)
Theoretische (pädagogische) Begründung des Ansatzes:	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung:	Gleicher Anteil (ca. 50 %)

Using Business Process Models to Foster Competencies in Requirements Engineering

Yvonne Sedelmaier & Dieter Landes
University of Applied Sciences and Arts, Coburg, Germany
{yvonne.sedelmaier, dieter.landes}@hs-coburg.de

Abstract

Requirements are of paramount importance for the quality of software systems. For various reasons, however, university students encounter difficulties in understanding the role of requirements and appropriately applying relevant methods to deal with requirements. This paper describes the concept for teaching requirements engineering that was devised at Coburg University of Applied Sciences. As a key idea, teaching requirements starts out from business process models. From these models, requirements for a workflow application can be derived and specified in a requirements document. A main benefit of this approach lies in the fact that requirements are not just presented as an abstract concept. Furthermore, students are exposed to the complexity of an almost realistic workflow application. Being more realistic than a toy project, the latter also improves understanding why requirements should be described precisely and provides opportunities to also exercise non-technical competencies that are important for successful requirements engineering.

1. Introduction

Software nowadays plays a central role in our lives: modern communication would be impossible without software, and even many seemingly non-IT products such as washing machines or automobiles rely increasingly on software. Software development, however, is complex and requires highly skilled individuals to cope with the task. In particular, many surveys such as the CHAOS studies [1] indicate that many software development projects fail. Often, a main cause for failures lies in the insufficient treatment of requirements. Therefore, education and training in requirements engineering is an important issue. Some possible approaches are discussed, e.g., in [2].

Yet, teaching requirements engineering is difficult due to various reasons that also surfaced in course evaluations. For one thing, students at universities often do not understand the role and impact of requirements. Since students are exposed to small programming assignments that they have to solve individually, they often do not see any point in paying account to requirements. These software development assignments primarily focus on the technical aspects of programming and specific programming languages. Therefore, the task that needs to be done is described precisely and originates from a domain that students are sufficiently familiar

978-1-4799-4970-0/14/$31.00 © 2014 IEEE CSEE&T 2014, Klagenfurt, Austria

with. Dealing with requirements seems to be a waste of time and fairly boring since the real fun is in programming.

As another consequence, students tend to take clear requirements for granted. They never faced the problem to elicit requirements from multiple groups of sometimes uncooperative stakeholders. They do not need to bother with conflicting and fuzzy requirements from an unfamiliar application domain.

Even if they are taught basic techniques of requirements engineering, they often suffer from the misconception that complexity scales up linearly: a single use case is fairly easy to specify, dozens of use cases are not. If the number of requirements grows, so do the interdependencies between them.

This paper describes the concept for teaching requirements engineering that we devised at Coburg University of Applied Sciences. As a key idea, teaching requirements starts out from business process models. From these models, requirements for a workflow application can be derived and specified in a requirements document. As main benefits, this approach avoids presenting requirements as an abstract concept and exposes students to the complexity of an almost realistic workflow application. Being more realistic than a small toy project, the latter also establishes a better understanding to what end requirements should be specified. Furthermore, a somewhat realistic setting also allows exercising non-technical competencies that are important for successful requirements engineering. In the next section, we will present our previous approach to teaching requirements engineering. We discuss the main weaknesses we identified, outline the goals we would like to achieve, and present the core elements of an improved teaching concept. We also provide the pedagogical underpinning of the changes we applied and outline how the approach was evaluated and which lessons we learned. Finally, we summarize our findings and point out some future extensions of our approach.

2. Teaching Goals

At Coburg University, requirements engineering is covered in a course called “Software Modelling” which is concerned with various aspects of requirements. In particular, this course views the treatment of requirements basically as a modelling issue. When students enter this course, they are already familiar with specifying functional requirements through use cases [Cockburn] [3]. In its initial format, the software modelling course started out with a short reiteration on use cases and continues on to alternative methods for specifying atomic functional requirements, such as the template-based approach of Rupp [4] [5]. Then, the focus of interest shifted towards non-functional requirements, addressing their relationship to quality attributes of software [6] and approaches to specify them, such as GQM [7] or PLanguage [8]. Furthermore, methods for requirements prioritization and recommended contents of requirements documents are discussed. As yet another major area, various approaches for business process modelling were covered since business process models constitute a major source of requirements, at least for workflow-oriented software systems. To this end, a particular scenario is modelled using notations such as Event-Driven Process Chains, BPNM, and Petri nets. Finally, the course explained how requirements act as the basis for estimating functional complexity and cost with methods such as Function Points [9] and COCOMO [10].

As a quite large practical exercise, students need to write a requirements document for an imaginary software system in teams of 5 to 7 members. This exercise aims at glueing most of

the theoretical contents of the course together since this task involved business process models, use cases, functional and non-functional requirements, priorities, and complexity estimation.

A revision of the teaching goals exhibited several shortcomings. As it turned out, the teaching goals for the software modelling course are on a somewhat abstract level:

- Students shall acquire a more tangible impression of the term “requirements”.
- Students shall understand the importance of requirements and shall be able to act accordingly.
- Students shall understand characteristic approaches to the specification of functional and non-functional requirements and their prioritization.
- Students shall understand the role of communication with other involved parties in requirements engineering.
- Students shall understand the role of business processes as a source of requirements.
- Students shall be able to collaboratively apply appropriate methods and notations in order to specify requirements for a sample software application. In particular, this encompasses the capability to write a requirements document, but also to understand a requirements document that the students did not write themselves.
- Students shall understand popular approaches to complexity and cost estimation for software systems.

These teaching goals primarily represent competencies to be fostered. The teaching goals are also specified in more detail in order to support course evaluation. For the sake of brevity, the detailed teaching goals are not included in this paper.

Unfortunately, students fell short of completely achieving these teaching goals. Rather, they had difficulties in really understanding the issues covered in the course. Course evaluations showed that they perceived the whole matter as fairly dry and theoretical and could not adequately value the role of requirements for their future professional career. In particular, they found writing requirements for small sample problems almost trivial. Furthermore, they were neither able to appreciate business models as input to requirements elicitation, nor could they properly understand the interrelationship between the various subjects in the course, but conceived them as a somewhat incoherent collection of topics. Therefore, we felt a need to revise several components of our teaching model. As a first step, we assessed several well-known pedagogical approaches for applicability in this context. In particular, we intend to put our approach on a sound theoretical pedagogical foundation. Details will be summarized next.

3. Pedagogical Underpinning

As a start, we clarified teaching and learning goals for ourselves and formulated them explicitly. On this basis, we developed a new didactical approach applying several elements from general didactics. The revised didactical approach should enable students to achieve the teaching goals and combines and adapts elements from several didactical theories.

In particular, we use required competencies as a basis to deduce teaching goals for this course. This approach follows the spirit of curriculum theory [11].

Then we analyzed the existing course structure and realized that the sequence of topics did not match our teaching goals well. Our main aim is that students should understand the importance of requirements for software engineering and how they can collect them. As a consequence, we restructured the course's contents and adapted the degree to which we emphasize specific topics. In particular, our teaching goals imply that emphasis should be put on business process models as a means to derive initial requirements instead of starting out with methods for describing requirements, no matter where they come from. Now, the basic question is how to first elicit and then describe requirements instead of simply describing requirements. Rather than teaching "stockpiling knowledge" we turned our teaching concept into an approach which first exposes students to a problem for which they need a solution. In such a situation, it is far easier for students to understand the necessity of the learning topics and figure out what this knowledge is necessary for. It is difficult for students to understand why they should learn specific things when they have neither place nor time to apply it or when they do not know what purpose the knowledge is intended to serve.

After explicitly expressing our teaching goals and restructuring the contents we needed to decide on teaching and learning methods. For this step we reverted to elements of the "learning-centered approach to general didactics" according to Heimann [12]. This theory focusses on the lecturers' perspective and tries to give them empirical data for planning a lesson. Thus, this approach builds upon empirical learning theories. According to this theory, a lecturer has to take decisions in four areas [13] (Berlin model of lesson planning), namely intention, theme and topics, methods, and media.

We also included elements from constructivist didactics by setting up a learning environment which enables students to adapt their individual mental worlds and learn to handle requirements by using business process modelling.

Problem-based learning is a didactical method which takes into account all aforementioned issues. It is a learner-centered approach and can be allocated to constructivism [14], [15]. Problem-based learning starts with a problem that students have to solve [16]. In order to cope with this challenge, students must recognize by themselves which further information and additional considerations are required to solve the problem. Thus, one key challenge is to collect lacking information which is needed to cope with the problem.

Problem-based learning is characterized by the complexity of the subject and an inquiry model. After students have been given a problem that needs to be solved, they organize their existing knowledge and identify areas where they need more knowledge [17]. In the next step they plan how to attain the required information and how to solve the problem. Bransford and Stein [18] summarize problem-based learning as follows:

- I = Identify problems and opportunities
- D = Define goals
- E = Explore possible strategies
- A = Anticipate outcomes and Act
- L = Look back and Learn

Going hand in hand with the changes in didactics from contents and topics towards outcome and competence-orientation, didactical approaches turned from direct instruction to activating learning styles such as project work [19] or problem-based learning.

4. Characteristics of the New Didactical Approach

Based on these aforementioned didactical considerations and our teaching goals we developed a new didactical approach which is characterized by the following aspects.

4.1. Restructuring of Contents

One important change of the didactical concept was a reorganization of contents. Based on the teaching goal to convey a deeper understanding of requirements in software engineering to students, we now emphasize process modelling as a way of extracting and describing requirements in software engineering. Modelling processes allows software engineers to get requirements indirectly from an organizational workflow instead of or in addition to asking future users which functional and non-functional features the new software should exhibit. As an additional obstacle, customers or product owners often cannot formulate or express their expectations towards the new software explicitly. In particular, non-functional requirements are quite difficult to describe. Examining steps in an organizational workflow and modelling them in a process model gives a high-level view of requirements. In additional, software engineers get a proper idea of the context in which the software will be used and can define interfaces more precisely. Therefore, we decided to emphasize process modelling and use it as the opening topic of the software modelling course.

4.2. Practical Exercises

To achieve a deeper understanding for the importance and difficulties of communicating requirements we developed two main exercises for students. Exercises used illustrative examples and included an exchange of process models between different student groups. First, students get the task to develop a process model for a scenario with which they are somewhat familiar. In our case, we reverted to a workflow for applying and claiming reimbursement of expenses for a business trip. Students get a leaflet which is used as a guideline by university members when they go on a business trip. It contains detailed rules for the application and reimbursement of a business trip and provides some details on how the process works. Students work in groups of four or five team members and extract distinct steps of the process before modelling them in a notation of their choice.

Communication in requirements engineering is a very important issue. Software engineers have to elicit requirements which are often expressed verbally or in writing by persons from different disciplines. To aggravate things, requirements are typically not expressed clearly and explicitly by stakeholders. Even using additional or predefined methods for requirements elicitation such as inquiry or observation techniques, software engineers never can be sure to understand exactly what future users really mean. This holds true even if software engineers record their findings in a wide-spread and commonly understood notation such as UML. Although the syntax is fixed, there is still plenty of room for semantic misunderstandings.

To give students a better understanding of these challenges, the process models from the first exercise are taken as a basis for the second one. Each student group is asked to review a

process model of another team. In class, they can ask clarification questions to the group from which the process model originated. Students realize quickly how difficult it is to model all necessary information into a process model and to understand what exactly the modelers wanted to express, even though the modelling notation is defined and all groups used the same guideline document as a starting point.

Finally, the groups need to write a requirements document on the basis of the process model they received from a peer group. Students learned before how a requirements document should look like and have access to an example requirements document. The requirements document should contain functional as well as non-functional requirements of the software.

In summary, to give students a deeper understanding of the necessity of requirements and the difficulties to describe them we use a real world problem and two exercises which reflect the world realistically. Using a real-life document is a good way to make students understand the degree of complexity of processes in an organization and a typical way in which processes are described, specifically in textual descriptions.

4.3. Shift from Instruction to Self-Organized Learning

During the term we slowly lead our students from a receptive way of learning through structured group work to an independent group work approach (see fig. 1).

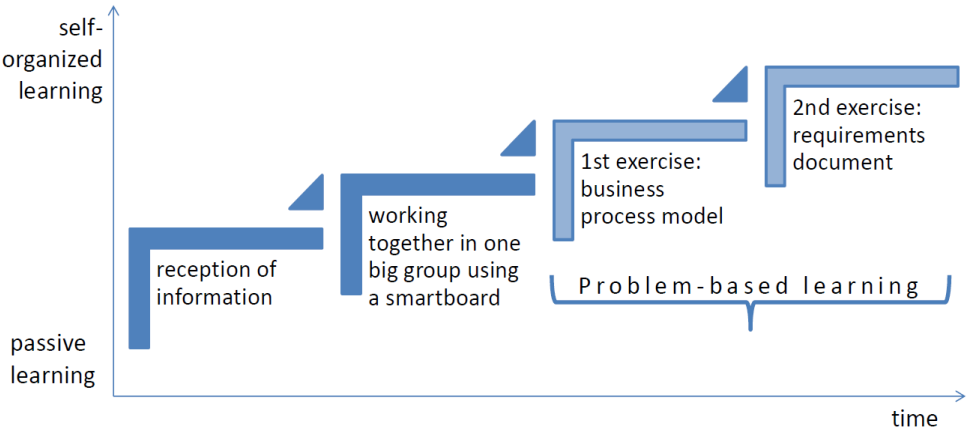


Figure 1: Shift from instruction to self-organized learning

Early in the term, notations for process modelling are taught in a receptive way. In a second phase, students need to develop a process model based on given instructions. To cope with this task they have to work together as a single big group of approximately 20 participants. We use a smartboard to activate all of them. A smartboard is an interactive whiteboard where students can use almost arbitrary modelling software and drag and drop model components with their hands like on a touchscreen. This approach fosters collaboration and is a first step towards learner-activation. Our experience shows that at the beginning of the lesson only a few students took part in the group work using a smartboard. Some students only watched others working and discussing, some needed time to get involved in the team process. At the end of the lesson, all students had contributed to the team result, although in varying extent. In a third step, we foster independence of our students' learning by means of group exercises. Group

work is an instance of self-directed learning approaches. Especially the first exercise is a problem-based learning approach and can be allocated to action-orientated didactical categories with focus on real-life experiences.

In our didactical approach students are prepared with basic information about modelling processes using several notations. In problem-based learning students have to apply their theoretical knowledge they learned earlier, e.g. on Petri nets or BPMN, and need to extend it by further details in a situation-specific manner as required. Then, the instructor acts as a coach rather than a teacher in the usual sense [20]. Problem-based learning as described in sec. 3 is one possibility to foster problem-solving skills and to make students understand the complexity of several topics, such as process modelling for requirements engineering.

5. Evaluation and Lessons Learned

All in all, our didactical approach worked pretty well and both students and lecturers learned a lot, while there is still room for some improvements.

One methodical approach was the mutual evaluation of process models. Students were asked to work together on the basis of another group's process model and review it. We asked students to issue at least three concerns with respect to the process model of their peer group. This process was structured by the groups themselves. As it turned out, students were not prepared well for a review process. On the one hand, they initially would see raised concerns as a personal criticism and tried to justify themselves. They were not able to see the opportunity to improve their process models and, indirectly, also their learning experiences. Students were not equipped with guidelines concerning which aspects of the process models should be paid particular attention. Yet, in the course evaluation at the end of the term it became clear that students after a while actually see the benefit of this method for identifying failures and in taking the perspective of another person – which was in fact the goal of this exercise.

In order to evaluate the course an evaluation scheme was used that contains items concerning the organization of the course, didactical aspects, and also BEvaKomp [21], a tool for the self-assessment of students' competencies. The evaluation that we performed shows that our students learned a lot, though at the expense of a heavy workload. Nonetheless, they can now understand the importance of requirements and present an overview about the main topics. But still, they feel that they are not able to write a requirements document by themselves or to assess the quality of a given requirements document (see fig. 2 and 3). All students answered they learned a lot in this course and all of them see the learning goals achieved.

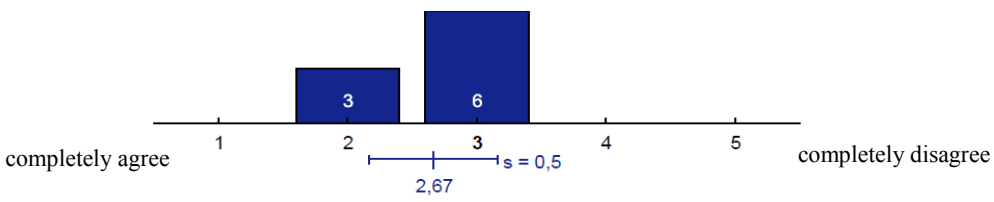


Figure 2: I feel able to write a requirements document.

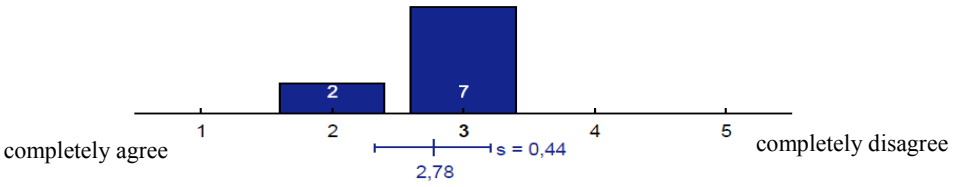


Figure 3: I feel able to assess the quality of a requirements document

The new structure of contents seems to be a good way to make students understand the overall software engineering context (see fig. 4). Also, group assignments were seen as a good method to give students a deeper understanding of requirements engineering (see fig. 5).

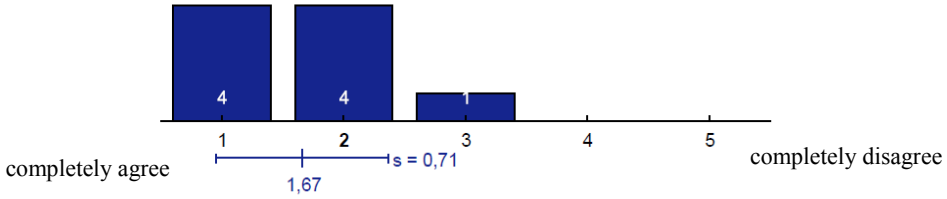


Figure 4: The contents' structure helps to understand the complexity of software engineering.

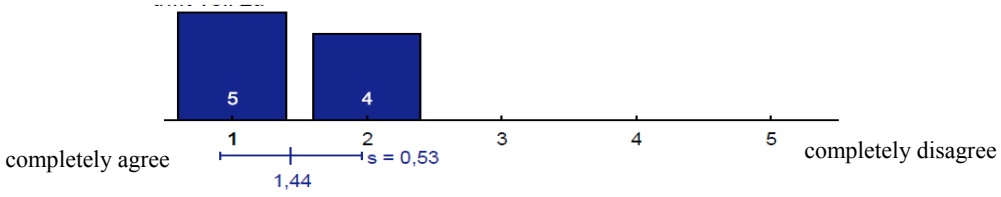


Figure 5: Exercises and group works supported the understanding of requirements.

6. Conclusion and Outlook

This paper outlines some rearrangements of the didactical concept of a requirements engineering course in order to improve students' understanding of the subject matter. As main changes, we restructured the sequence of topics to a more coherent one, introduced practical

group assignments, and gave room to self-directed learning. Thus, we combined elements from curriculum theory, learning-oriented didactics, and constructivist didactics.

Although the changes in our teaching concept seemed to work out well, there is still room for further improvement. For one thing, the sample of students involved in the evaluation was fairly small, thus posing a potential threat to validity. In order to put our findings on a firmer basis, this sample needs to be enlarged in future offerings of the course.

In addition, the didactical concept might be refined with aspects such as quality assurance through formal requirements reviews. This might be achieved by reviewing an existing requirements document as a preparation for actively writing such a document. As an expected benefit, students are arguably better aware of the shortcomings they find in the reviewed requirements document and consequently better prepared to avoid these shortcomings in their own requirements specification. Alternatively, each group of students might review the document of another team. This might lead to more intense and accessible feedback since feedback is currently only provided by the instructors, but not by their peers.

In upcoming offerings of the course, students should be given more concrete and detailed instructions for the review process. For instance, it would be helpful if there were guidelines as to which aspects to consider in the review, e.g. formal aspects of the process model or steps they do not understand in the model. Since typical students are only marginally experienced in reviewing the work of others, it seems to be desirable to hand out more information about constructive feedback. This didactical approach is expected to improve students' personal skills, namely their capability to give and obtain feedback without feeling accused.

Another way to structure the learning process would be "learning by example". We might use a first document review to present best practices. Students might then follow these best practices as a template in a second review when they try to imitate the lecturers' approach.

Occasionally, students pointed out that consequences of "bad" requirements would be immediately tangible if the system were not just specified, but also implemented. This is certainly a valid point, but implementation of a non-trivial application can hardly be incorporated into the software modelling course due to lack of time. Rather, this is deferred to an additional elective course which broadens the scope to the realization of a complete capstone project, from requirements elicitation and specification to implementation, test, and hand-over [22], but also pays attention to non-technical skills that are necessary to render a software project successful.

Next steps in research are directed towards a framework for analyzing and measuring competencies. Thus, conclusions on the effectiveness of didactical approaches such as the one we devised for the software modelling course can be drawn. Typically, it is assumed that particular competencies are fostered if students achieve the teaching goals. This hypothesis might be evaluated more objectively if there were a tool for competency assessment. Evolution of competencies can be made transparent by applying this tool at the course's beginning and end.

Acknowledgment

The research project EVELIN is funded by the German Ministry of Education and Research (Bundesministerium für Bildung und Forschung) under grant no. 01PL12022A.

References

[1] J. L. Eveleens and C. Verhoef, “The rise and fall of the Chaos report figures,” *IEEE Softw,* vol. 27, no. 1, pp. 30–36, 2010.

[2] A. Herrmann, A. Hoffmann, D. Landes, and R. Weißbach, “Experience-oriented Approaches for Teaching and Training Requirements Engineering: An Experience Report,” in 20th Int. Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ14), Cham (Schweiz): Springer, 2014, pp. 254–267.

[3] A. Cockburn, *Writing effective use cases*. Boston: Addison-Wesley, 2001.

[4] C. Rupp, *Requirements-Engineering und -Management:* Professionelle iterative Anforderungsanalyse für die Praxis, 5th ed. München: Hanser, 2009.

[5] K. Pohl and C. Rupp, *Requirements engineering fundamentals:* A Study Guide for the Certified Professional for Requirements Engineering Exam, 1st ed. Santa Barbara Calif: Rocky Nook, 2011.

[6] *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models*, 25010:2011, 2011.

[7] R. van Solingen and E. Berghout, *The goal/question/metric method:* A practical guide for quality improvement of software development. London, Chicago: McGraw-Hill, 1999.

[8] T. Gilb and L. Brodie, *Competitive engineering:* A handbook for systems engineering requirements engineering, and software engineering using planguage. Burlington: Elsevier Butterwoth-Heinemann, 2005.

[9] D. Garmus and D. Herron, *Function point analysis:* Measurement practices for successful software projects. Boston: Addison-Wesley, 2001.

[10] B. W. Boehm, *Software cost estimation with Cocomo II*. Upper Saddle River: Prentice Hall, 2009.

[11] C. Möller, *Technik der Lernplanung:* Methoden und Probleme der Lernzielerstellung, 3rd ed. Weinheim: Beltz, 1971.

[12] P. Heimann, *Unterricht,* 8th ed. Hannover: Schroedel, 1976.

[13] W. Schulz, “Unterricht - Analyse und Planung,” in *Unterricht: Analyse und Planung*, P. Heimann, G. Otto, and W. Schulz, Eds, Hannover u.a: Schroedel, 1965, pp. 13–47.

[14] S. Kemp, *Constructivism and Problem-Based Learning.* Available: http://www.tp.edu.sg/pbl_sandra_joy_kemp.pdf (2013, Oct. 09).

[15] J. R. Savery and T. M. Duffy, “Problem Based Learning: An Instructional Model and Its Constructivist Framework,” in *Constructivist learning environments: case studies in instructional design*, B. G. Wilson, Ed. 2nd ed, Englewood Cliffs N.J: Educational Technology Publications, 1998, pp. 135–148.

[16] H. S. Barrows and R. M. Tamblyn, *Problem-based learning:* An approach to medical education. New York: Springer, 1980.

[17] M. Savin-Baden, *Problem-based learning in higher education:* Untold stories. Buckingham: Society for Research into Higher Education & Open University Press, 2001.

[18] J. D. Bransford and B. S. Stein, *The ideal problem solver:* A guide for improving thinking, learning, and creativity, 2nd ed. New York: W.H. Freeman, 1993.

[19] E. Terhart, *Didaktik:* Eine Einführung. Stuttgart: Reclam, 2009.

[20] M. S. Knowles, E. F. Holton, and R. A. Swanson, *The adult learner:* The definitive classic in adult education and human resource development, 7th ed. Amsterdam, Boston: Elsevier, 2011.

[21] E. Braun, *Das Berliner Evaluationsinstrument für selbsteingeschätzte studentische Kompetenzen (BEvaKomp)*. Göttingen: V & R Unipress, 2008.

[22] D. Landes, Y. Sedelmaier, V. Pfeiffer, J. Mottok, and G. Hagel, “Learning and teaching software process models,” in Global Engineering Education Conference (EDUCON), 2012, pp. 1153–1160.

Anhang 1.4

Sedelmaier, Y. & Landes, D. (2015). Active and Inductive Learning in Software Engineering Education. In 37th International Conference on Software Engineering (ICSE) (S. 418–427) © 2015 IEEE/ACM. Reprinted, with permission, from Sedelmaier, Y. & Landes, D., Active and Inductive Learning in Software Engineering Education, IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015

Impact-Factor (falls vorhanden): A*

Eigenanteil:

Hauptautorin;

Konzeption des beschriebenen Ansatzes:	Gleicher Anteil (ca. 50 %)
Theoretische (pädagogische) Begründung des Ansatzes:	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung:	Gleicher Anteil (ca. 50 %)

2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

Active and Inductive Learning in Software Engineering Education

Yvonne Sedelmaier, Dieter Landes
Faculty of Electrical Engineering and Informatics
University of Applied Sciences and Arts
96450 Coburg, Germany
{ yvonne.sedelmaier, dieter.landes }@hs-coburg.de

***Abstract*—If software engineering education is done in a traditional lecture-oriented style students have no other choice than believing that the solutions they are told actually work for a problem that they never encountered themselves. In order to overcome this problem, this paper describes an approach which allows students to better understand why software engineering and several of its core methods and techniques are needed, thus preparing them better for their professional life. This approach builds on active and inductive learning. Exercises that make students actively discover relevant software engineering issues are described in detail together with their pedagogical underpinning.**

***Index Terms*—software engineering education; didactical approach; inductive learning; active learning; higher education;**

I. Introduction

Learning and teaching software engineering at universities is difficult. Part of this difficulty lies in several misconceptions that are common among students. Traditionally, education in software development focuses on programming, i.e. students initially try to master technical difficulties in terms of getting used to programming using a particular programming language like Java, C, or C++. These programming assignments, however, do not mirror real-life software development tasks realistically since they are precisely specified and supposed to be solved in a one-person effort. In contrast, real software projects are a multi-person effort, requiring a lot of coordination among stakeholders as well as among team members. Furthermore, requirements tend to be unclear for quite some time, in particular if real external customers with different cultural backgrounds are involved.

Students successfully mastered programming-in-the-small before being exposed to software engineering methods and techniques which aim at coping with complex projects. Students frequently assume that their approach, which worked fine on small assignments, scales up easily to more complex tasks. Therefore, software engineering seems to be fairly irrelevant, and many of the offered methods and techniques are fairly abstract and seem to be close to common sense without a convincing motivation for their use.

Exposing students to the task of developing a complex piece of software in a team right away is not a viable alternative since students are lacking elementary programming skills.

This paper presents an approach to teaching and learning software engineering that builds on ideas from active and inductive learning [1]. In order to arrive at a better understanding of why there is a need for software engineering and of what software engineering actually is.

This approach has been applied to an introductory course on software engineering. In the remainder of the paper, the general constraints of this course will be outlined together with some of the intended learning outcomes. Then, the core constituents of the didactical approach to achieve these learning outcomes will be presented along with their pedagogical underpinning, and observations will be discussed.

II. General Set-Up

A. Characteristics of the Course

In their first year, students in the bachelor program of informatics at Coburg University of Applied Sciences are exposed to various basic concepts of informatics. In particular, they learn Java programming. Practical assignments are targeted towards a general understanding of fundamental concepts of computer programming and getting hands-on experience with basic and more advanced features of Java.

In the third semester, there is a shift from programming-in-the-small to programming-in-the-large. To that end, students need to take a compulsory introductory course in software engineering. This software engineering course has 4 contact hours per week. Typically, some 40 students are enrolled in this course.

B. Intended Learning Outcomes

The most important intended learning outcomes of the software engineering course are:

- Students shall understand the need for a principled, engineering-like approach to developing complex software systems.
- Students shall acquire an understanding of the areas that in their entirety constitute software engineering.

978-1-4799-1934-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICSE.2015.174

- Students shall understand the fundamental phases of software engineering.
- Students shall understand how activities in software engineering may be arranged into a well-known process model, including pros and cons of these process models.
- Students shall be able to understand the purpose of selected widespread methods, techniques, and notations.
- Students shall be able to apply selected methods, techniques, and notations in the context of moderately complex software projects.
- Students shall exercise and enhance non-technical skills such as working techniques, self-responsibility, collaboration, ability to abstract, or presentation skills.

The latter non-technical skills will not be exercised in isolation, but rather as a by-product when the former intended learning outcomes will be addressed. This is necessary since many of those skills are context-sensitive in the sense that they get a very specific flavor depending on the domain in which they need to be exhibited [2]. For instance, collaboration in teams is different in a software engineering context than, say, in civil engineering. Non-technical skills shall be gradually enhanced to prepare students for a complete capstone software development project [3] in their final year of study.

III. Didactical Approach

Core elements of our didactical approach will be described using three facets. First, the didactical method as such is described. Second, the intended learning outcomes related to the respective element of our approach are explained. Third, observations when applying this specific method are discussed. This way of describing our approach has been chosen because each method is adapted at any point of time according to the students' requirements and perceived needs. This way of adapting didactical approaches to students' requirements is called participant orientation (reference) or "reading and flexing" [4]. So, we try to adapt the course to our students, considering their previous knowledge and interests.

A. Course Introduction and Motivation

1) Goals

Since one of the learning outcomes aims at understanding the need for a principled approach to software development, the first activity targets the motivation for software engineering. Basically, we decided to start the course by picking the students up at the point where they actually are, namely at small programming tasks which they can solve on their own, and exposing them to the limits of their hitherto existing approach. To that end, we started with a small "programming" assignment they should solve individually.

2) Course of Action

To commence with, students were asked to devise a program for a simple numerical problem which, at first sight, was specified precisely. In particular, the assignment read as follows:

"Exercise 1a: Develop (individually or in teams of two) a software program which determines whether a natural number, which was given as input, is a prime number or not. As a reminder: prime numbers are divisible without remainder only by 1 and themselves. Working time: 15 minutes."

When students were done with this task, they were given a second task:

"Exercise 1b: A university in the northern part of Bavaria has a pool of vehicles at its disposal, which can be used for business trips. Since that particular university is fairly innovative, it would like to employ slightly more up to date measures to administer its pool of vehicles than just paper / calendar and pencil.

Support the university in this undertaking by developing, in teams of four, a software program for managing the university's pool of vehicles. Working time: 60 minutes."

The second assignment was deliberately designed in such a way that students were bound to fail for various reasons. On the one hand, the description of what the vehicle management software was supposed to do was way too fuzzy; on the other hand such a system cannot be implemented within an hour in any case.

Whenever a team noticed that they were not in a position to solve the task, exercise 1b was "substituted" by a third assignment to clarify the differences between the two initial ones. The third assignment read as follows:

"Exercise 1c: You just worked on two tasks: programming a prime number checker and developing software for vehicle pool administration. Arguably, everyone was able to solve the first task smoothly, while the second one constituted a major challenge.

Think about why this is the case.

- What are the essential differences between the prime number checker and the vehicle pool administration system?
- How would you explain to your customer why you cannot develop such a piece of software within an hour under the given circumstances?
- Which tasks would you need to solve if you would want to develop such a vehicle pool administration system successfully in teams of four persons?

Record your results and give a short plenary presentation of your results. Working time: 30 minutes".

Due to time restrictions, writing cards individually with respect to the three questions in exercise 1c substituted plenary presentations.

These cards were collected and clustered by the instructor, together with comments or rectifications where appropriate.

3) Observations

Even though students did not actually implement a running program in Java, but rather devised the algorithm in Java notation, they were able to solve exercise 1a smoothly within the given time. Since they had worked on various assignments involving prime numbers, the given task was fairly familiar to

them. One volunteer student presented his solution, while other commented and improved on it. Still, students did not yet realize that the presented solution only covered the most common case, but neglected boundary situations, namely the fact that 1 is no prime number. Although the assignment seemed to be clear and precise at a quick glance, this special case was deliberately missing in order to point students to the issue of implicit or missing requirements and systematic tests with boundary cases at a later point of time.

Concerning exercise 1b, the situation was more diverse across teams. Some teams started work on the assignment intensively, though asking the instructor for more detailed requirements. Once they got additional information, they worked on building a use case model from the requirements that they had been given. They did so even though they implicitly knew that modeling use cases would use up the available time of one hour almost completely, leaving no time to build the system as mentioned in the assignment.

Other teams believed if they only knew how to deal with a calendar, they would be able to cope with this task and implement the system within an hour. Apparently, these teams faced a gross misconception with respect to the real effort of building the vehicle management system. Furthermore, these teams jumped at an alleged solution without giving a second thought what the problem really was.

Some other teams were too confused to really start working on the exercise. As mentioned above, this kind of confusion was provoked with intent: teams should realize that there are fundamental differences between small programming tasks and software engineering. Thus, the complexity of real software engineering should be pointed out.

Exercise 1c was handed out to all the teams sooner or later. The teams that hesitated to start due to their confusion what to do were given the "alternative" assignment right after they expressed their uneasiness. The instructor questioned the solution-minded teams if they really believed they would cope with the task even after they solved their calendar problem. After some doubt on this issue had been provoked, the teams were given exercise 1c. Finally, the instructor stopped the requirements-oriented teams after they finished a rough sketch of the use case model and reassigned with exercise 1c.

Students recognized that software engineering is more than programming. Rather than being told by the instructor, students discovered on their own that additional issues such as project management, communication, team aspects, requirements engineering, cost and time estimation, and testing need to be considered. Figure 1 shows some of the results of exercise 1c after the instructor had clustered the cards which students had handed in:

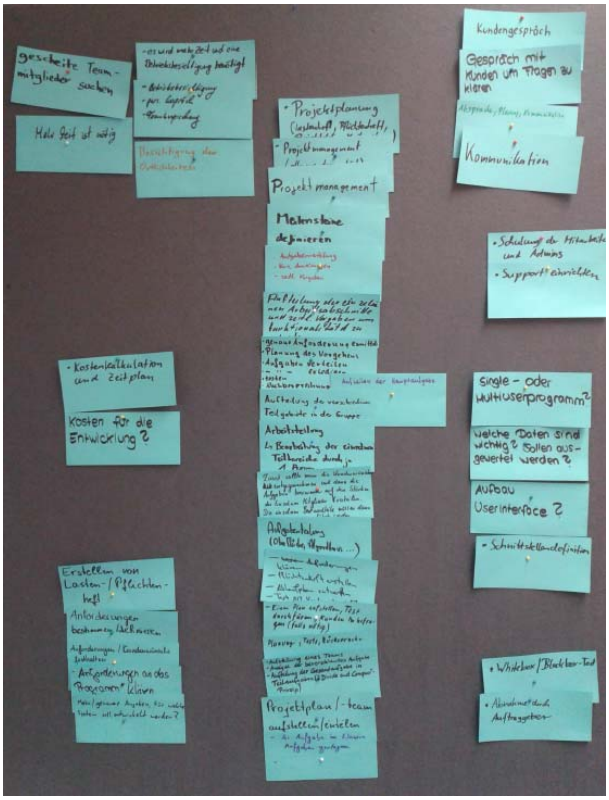


Fig. 1. Results of Exercise 1c

Clustering and commenting on the cards that students had written generated an intense discussion with lively participation of almost all students in the classroom. One student even praised this class as "the most instructive and funniest one during his studies so far".

4) Didactical Aspects

Instructors can refer back to the results of this group work during the whole term when introducing a new topic. This helps to create a big picture of software engineering and exposes the interrelationships among software engineering topics as well as the benefit of the issues addressed in the following months. Through working on this exercise students developed special interest in the topics they brought into discussion by themselves. In a sense, their learning is more self-directed since they were no longer discussing only topics that the lecturer brings in, but rather issues of their own interest. Learning takes place when students accept topics as relevant for their purposes [5]. As a consequence, they are interested in the issues and motivation for learning arises. Instead of teaching solutions for problems which students cannot even imagine, we make them see and understand the problems right from the beginning.

The course introduction follows the didactical principle of inductive learning, i.e. learning starts with students recognizing a problem that arises from a given task for example. Instead of

teaching solutions for problems which students never had so far students should first understand which possible problems might exist. Inductive learning is also called “teaching backwards” because it does not start from a solution, but rather from a problem. This guides students to a deeper understanding for the learning contents and increases motivation to engage in the learning topics.

After recognizing the problem they learn about possible ways to solve it and apply their newly acquired knowledge (learning by doing). Educational psychology considers these principles as main factors for successful learning [5]. Instructors did neither want to teach solutions for problems students never faced before, nor should students pile up theoretical knowledge that they forget right after examination since they see no point in remembering them for their future career.

So, we want to make students think about problems to open their minds for possible solutions. This is perfectly in line with constructivist didactics. Von Glasersfeld, for example, explains: "There is no infallible method of teaching conceptual thought, but one which has most success is to present to students with situations in which their usual way of thinking fails." [6]. This is exactly what we wanted to achieve and why we started with such a three-step exercise.

B. Areas of Software Engineering

After compiling an initial collection of relevant issues for software engineering, the results that students came up with were compared to and contrasted with well-known software engineering “guidelines” such as SWEBOK [7] and CMMi [8]. Both, SWEBOK and CMMi, highlight important areas for successful software engineering.

1) Goals

This additional assignment primarily served the purpose to refine the results of exercise 1c, i.e. deepen the understanding of what constitutes software engineering. Moreover, the instructors wanted to learn more about students’ ability to abstract as well as their work techniques through observing students working on this assignment. Students should skim through large texts [7] [8] and extract their main messages. Furthermore, instructors wanted to figure out what students already knew about more advanced issues such as assessment models for software engineering capabilities.

2) Course of Action

In order to observe working techniques and group roles students were split into two relatively large groups of 10 to 14 persons each. They were given the following task:

“Exercise 2: Which other areas that are relevant for software engineering can you identify in the documents „Software Engineering Body of Knowledge“ (SWEBOK) [Group A] or „Capability Maturity Model Integration“ (CMMi) [Group B], respectively?

- Note your results on yellow [Group A] or white [Group B] metaplan cards, respectively.
- Present your results to the class and explain briefly why the subject seems to be relevant from your point of view.

Working time: 20 minutes”

Students did receive no further guidance on how to work together or how to organize themselves as a group. The only clear assignment concerned which result was expected in the end.

3) Observations

Instructors observed very heterogeneous working techniques and competencies to collaborate and interact within a team. One group worked together quite closely, discussed with each other, and merged their partial results to a single team result (see left side of fig. 2).

Fig. 2. Results of Exercise 2

This group included several students with prior work experience. These persons organized the team, played a leading role, acted self-responsibly and also took care for the team result.

The second group did not work together at all. This group disassembled into various small subgroups of two to three persons, which worked on the assignment in a fairly isolated manner. When it came down to present their results, the subgroups did not merge their findings, but rather one of the subgroups presented their individual results. Consequently, the second group delivered results that were only moderately useful (see right side of fig.2).

Group results indicated that neither of the teams ever came across assessment models for software engineering capabilities before, but also failed to grasp the idea behind CMMi from

skimming through the document. Furthermore, several process or knowledge areas that were mentioned in the texts were completely alien to students, which resulted in misconceptions of what these issues were all about. Configuration management, for instance, was explained as "the discipline that is responsible for the parameters that can be adjusted to adapt the final result to specific needs". These observations indicate that student still have difficulties in distilling the core message from large texts, given only a short period of time for reading over the text.

4) Didactical Aspects: Follow-up at the Meta-level

After observing heterogeneous ways of collaboration and very different levels of team competence instructors decided to feed their impressions back directly and make the intended learning outcomes explicit. Instructors explained why these kinds of interactive didactical approaches had been chosen and what students were expected to learn.

In doing so, instructors referred back to the results of exercise 1 where students themselves had identified the need for collaboration and cooperation, and the importance of communication in software engineering processes. These soft skills are indeed necessary for software engineering [2] and should be addressed in university education. Instructors pointed out that in the current course students are given opportunities to foster exactly theses competencies in a protected space without worrying about consequences that they might face in real business life.

Students should not only learn about communication and working together theoretically, but also apply and foster these competencies in practice. Making these intended learning outcomes explicit triggers students' learning and thinking processes.

C. Software Process Models

1) Goals

At this point, students had some understanding of what is necessary for developing complex software systems successfully. The next exercise was targeting the issue of how various activities can reasonable be arranged on a time scale, i.e. assembled in a software process model. In essence, students should understand that software engineering is a creative process that can be carried out following the process model that is best suited to the particular situation – there is no single best process model.

As a by-product, work techniques and (self-) responsibility should be fostered through active learning styles.

2) Course of Action

Students were invited to ally in groups of approximately five members in order to work on a specific software process model. Each group was allowed to choose the process model that they would want to work on. In particular, six software process models were to be examined in more detail, namely

- Waterfall model and V model,
- V model XT,
- Spiral model and incremental software development,
- Unified process,
- Extreme programming, and
- Scrum.

Each student group received a short extract of 3-5 pages from two textbooks [9] [10] which characterized the essence of a specific software process model. In the assignment, students were supposed to identify the main characteristics of a software process model and to present their findings to the class:

„Exercise 3: Organize yourselves in groups of approximately five members. Read the text on the process model of your choice. Identify the core aspects of the process model and prepare them in such a way that you can explain the process model to your fellow students. In particular, you are asked to pay attention to the following aspects:

- What are the core features of the software process model?
- Which roles are there and what are they responsible for?
- What are the pros and cons of the process model?
- For which type of project is this process model (in)appropriate?

Think about how you might best prepare and present your findings to your fellow students. Your results should be available in a format that can be uploaded to Moodle.

Working time: 30 minutes plus presentation: 10 minutes."

3) Observations

Instructors noticed that the self-directed division into groups and the selection of the process model worked quite well, with the exception of a "group" that consisted of a single student. Since this had the effect that all six process models on the list could be covered, instructors did not force the lone student into one of the other groups.

The performance of the groups gradually tended to become more homogenous. This became visible in the results they gained. Obviously, students enjoyed working actively and presenting their results. All teams, even the singleton team, delivered acceptable results and presented them to others in an entertaining way.

Five out of six teams used PowerPoint for their presentation and acted as a team during the group work process. Also, most of the presentations were jointly given by three members of the team, which also indicates some degree of organization within the team.

In effect, everybody gained an overview of all the software process models on the list, even though some aspects still were not mentioned. Some of these aspects were added by one of the instructors when commenting on the respective presentation.

4) Didactical Aspects

This didactical approach is called "learning by teaching" [11] [12] and helps students to learn how to structure and organize their working processes. Students are responsible for the results of their work on which possibly others depend. Therefore, this approach puts some implicit pressure on

students: if one group provides weak or unusable results, their fellow students will consequently have to spend more effort to obtain information on this topic because they have to read the respective texts in addition to their own task. This is a step towards problem-based learning [13] [14].

D. Software Process Models Revisited

1) Goals

Another learning cycle was added in order to deepen the understanding of software process models and to further clarify their differences. Ideally, every single student gets involved and gets a feedback with respect to her current status of knowledge on process models.

2) Course of Action

Exercise 4 consists of 15 statements about particular process models namely a) waterfall model / V model (XT), b) Unified Process, and c) Agile methods (i.e. Scrum and Extreme Programming). Students are required to indicate if a statement is true or false. Originally, the collection of statements had been part of the final examination of the previous course one year ago.

Students entered their decision (either true or false) on each of the statements on paper. The instructors collected the answer sheets and aggregated the results in simple histograms which indicate for each statement how many students voted for true or false, respectively.

In the following lesson, the answer sheets were returned to the students in their original form, i.e. without correction. The instructors went through the statements again, along with the votes for true or false. For questions without a clear majority or a majority for the incorrect answer, instructors initiated a discussion that involved justifications for their answers from both camps, and the vote (by show of hands) was repeated in order to identify a potential turnaround of opinions. Finally, instructors revealed the correct answer with a short explanation where needed.

3) Observations

Exercise 4 showed a broad majority for the correct answer for most statements, which indicates that the previous exercise 3 had been quite successful in generating an understanding of the essentials of the chosen process models.

The discussions about the less obvious statements clarified matters that had not been understood well enough.

As a by-product, each student got an immediate feedback on how good she did on the test by contrasting her answers to the correct ones.

4) Didactical Aspects

Our approach to teaching software process models consists of a two-step approach: First, learning by teaching (see sec. III.C) and, second, deepening the understanding of software process models by answering questions. In the first step, students should think about the statements for themselves before they were asked to explain their decision to other students, in particular those with a different opinion.

This didactical approach contains core elements of "peer instruction", a method which was developed by Eric Mazur [15]: A theoretical input aims at understanding basic concepts, underlying principles, and ideas of the topic. Then, students are given a feedback whether they understood these principles or if there are misconceptions which must be eliminated. This happens by justifying their decisions to another student and listening to a different opinion. Mazur argues that peer instruction helps students to eliminate misconceptions and deepen their understanding. As an additional benefit, peer instruction encourages students to get actively engaged in the learning process. In particular, this is the case even for students which are quite passive otherwise [15]. This is perfectly in line with our idea of active learning.

Our approach follows the idea that repetition supports learning [16]. Repetition fosters remembering the learned contents and helps to transfer it from short-term-memory to long-term-memory. Furthermore, trying to make sense of given statements about software process models deepens the understanding for the topic and adds some additional information. Attaching a meaning to the learned things is a precondition for remembering them [17].

E. Requirements Elicitation as a Communication Process

After software process models have been covered, the focus of the course shifted towards a more detailed look into one of the life-cycle phases, namely requirements analysis.

1) Goals

In our experience, novice students suffer from the misconception that requirements for software systems can be collected almost without effort, e.g. since customers provide them "on a silver tray". In reality, requirements elicitation is a communication process that is also heavily influenced by the notations that are used for the documentation of requirements. Exercise 5 aims at making this clear to students through a hands-on experience.

2) Course of Action

Students are divided in groups of four persons each. Within each group, there are two roles, namely "writers" and "painters", which are taken by two persons each.

For the exercise, the painters leave the classroom and move to another room. The writers stay in the classroom and are shown an abstract drawing like the one in figure 3.

The writers got 5 minutes to describe the drawing they saw in textual form – sketches are not allowed.

After 5 minutes, all descriptions were collected and handed out to the respective painters of each team in the other room. The transfer of the descriptions was done in such a way that writers and painters do not get in touch.

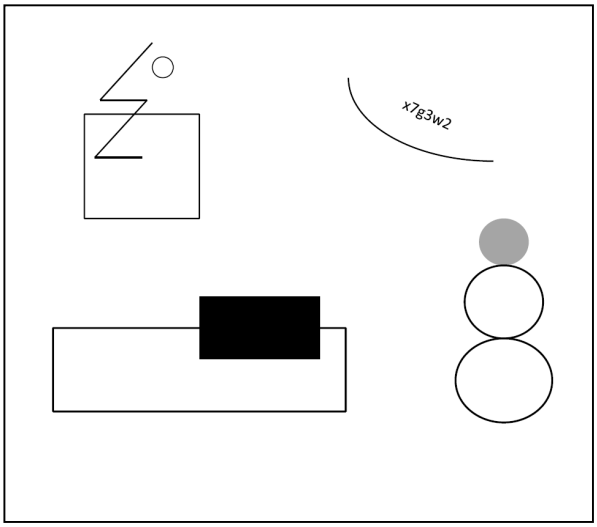


Fig. 3. Exercise 5

When the painters got the descriptions that their teammates generated, they had 10 minutes to reproduce the original drawing from that description as faithfully as possible.

3) Observations

This exercise resulted in drawings that typically deviate considerably from the original one. Often, the painters in the group misunderstood parts of the description they received from the writers (see fig.4).

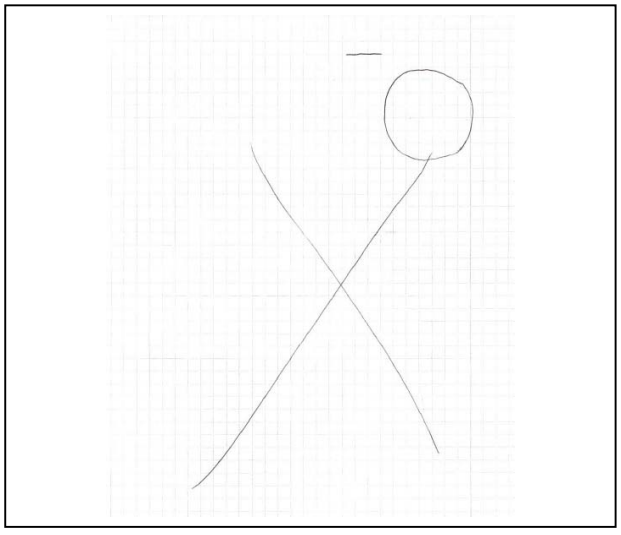


Fig. 4. Result of Exercise 5 (Example)

Some teams did better than others, especially if they realized that they can split the drawing into four quadrants and share the job of describing the drawing between the two writers by concentrating on two of the quadrants each. In essence, the same holds for the painters. Also, those teams that used metaphors (a part of the drawing resembles a snowman) generally produced better results.

When asked, students stated as main insights that they took away from this exercise that they now understand how hard it is to communicate information faithfully, that implicit assumptions are not shared globally, omissions and misunderstandings are common, and text is not appropriate to describe information that is to the most part pictorial.

4) Didactical Aspects

This approach again follows the idea of inductive learning as described in sec. III A.4. Again, students are confronted with a problem before they are given possible solutions. This is done in order to arouse students' problem awareness.

F. Additional Activities

In addition to the exercises mentioned in the previous sections, more exercises have been carried out that follow the same spirit. In particular, requirements elicitation methods and the most common UML diagram types were covered in essentially the same format that had been used for process models (see sec. III.C).

In addition, one exercise confronted students with two relatively complex UML diagrams, namely a (hierarchical) state diagram and a sequence diagram. Students were asked to summarize the situation that was modelled in the diagrams, thus forcing them to make sense of the diagrams and rephrase their interpretation of the diagrams in prose. Students were also asked to identify notational elements that they did not understand precisely in order to stimulate a discussion of the whole group.

In a lab exercise over a period of 4 weeks, students worked on a case study, namely a (hypothetical) system for the management of the university's car pool. In groups of 2 or 3 persons, students developed typical UML models to describe the setting of the case study faithfully.

These exercises and assignments again follow the idea of engaging students actively in order to stimulate their learning. Yet, the latter exercises cannot be discussed in greater depth due to space limitations.

IV. Discussion & Evaluation

The described didactical approach leads students through several phases (see fig. 5).

All in all, it grounds on an inductive learning style [1]. Inductive learning can also be described as "teaching backwards". It is the opposite of deductive teaching which starts with fundamentals, derives some general statements, and then provides some examples before working on some homework, labs, or projects. In contrast to that traditional way of teaching inductive learning starts from the other side: At the beginning, challenges, problems, specific questions, or observations are given to the students. This is the basis for defining a problem or need before the instructor explains prin-

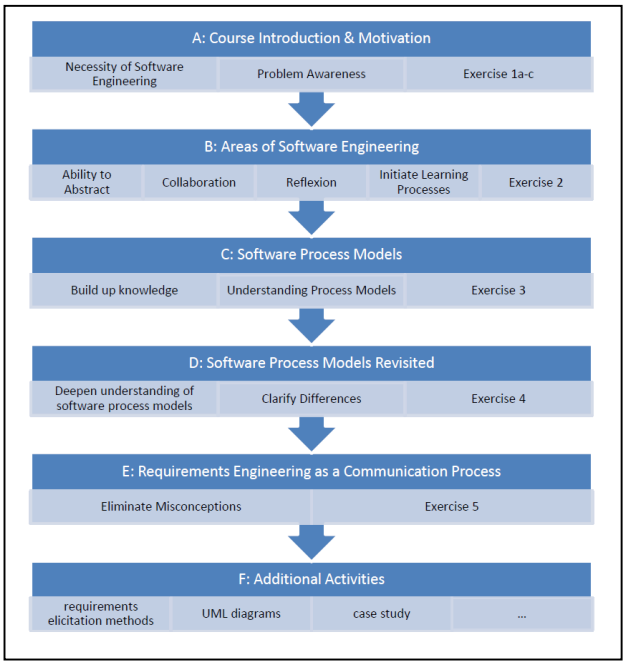


Fig. 5. Course Design and Intended Learning Outcomes

ciples and methods. Then some solutions are identified or refined. This approach also helps correcting misconceptions as described above, and supports arousing problem awareness.

This didactical approach starts with making intended learning outcomes explicit and then developing didactical methods according to students' needs. Therefore, microdidactical elements are adapted dynamically during the course.

Adapting microdidactical elements during the lesson is based on the didactical principle of participant orientation ("Teilnehmerorientierung") [18] [19] which is perfectly in line with constructivist didactics. Hunt [4] describes it as "reading" and „flexing": Reading means attentively observing students, while flexing concerns reacting on recognized requirements and needs. This generates an iterative process of adapting teaching and learning.

Several times, we received positive feedback from our students concerning this inductive didactical approach right after the lessons. As we observe, nearly all of them take active part in the given exercises; moreover, most of them seem to have fun to work at the assignments. This is also reflected in the observation that none of the students dropped out during the term, even though attendance was not compulsory. All in all, this indicates a quite high motivation and the impression that attending the course is worthwhile.

We also observe that students got closer to the intended learning outcomes step by step. They improve their work techniques and presentation skills by coping with each exercise. To back up our observations and to get some additional valid data we applied SECAT (Software Engineering Competency Assessment Tool) which evaluates quality of teaching by assessing students' competencies [20].

SECAT is based on self-estimations of students' competencies taking technical as well as non-technical skills into account which also reflect the intended learning outcomes. Further developed competencies indicate a good didactical approach. In addition, students' views are complemented with instructors' assessments. SECAT allows to measure students' competencies not only by assessing technical knowledge but also by assessing non-technical context-sensitive skills [2].

SECAT allows focusing on the assessment of one or more of nine criteria which are allocated to three levels of competence (see fig.6).

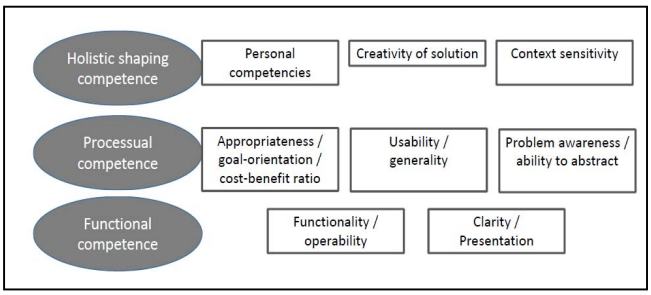


Fig. 6. Levels of Competence in SECAT

Non-technical skills are high-level competencies in contrast to functional technical knowledge or the ability to present some content. In this case, our intended learning outcomes (see sec. II.B) imply a focus on the three criteria functional competence, problem awareness, and personal skills. Our teaching goals, namely improving problem awareness with respect to the several aspects of software engineering, fostering communication, presentation and work organization skills, and strengthening self-reflection as a basis of competence development are reflected in these three criteria. Each of the three criteria is assessed by several questions. Most of the questions refer to holistic shaping competence (see fig. 7).

SECAT was employed at the end of the term. Table 1 shows the results of students' self-assessment. Each item was rated on a scale from 1 (not present / no improvement) to 4 (definitively present / high improvement). Items corresponding to the same competence are averaged. Thus, good ratings of most items indicate a large improvement of the respective competency. Colors in the table express the degree of competence improvement – the darker, the better.

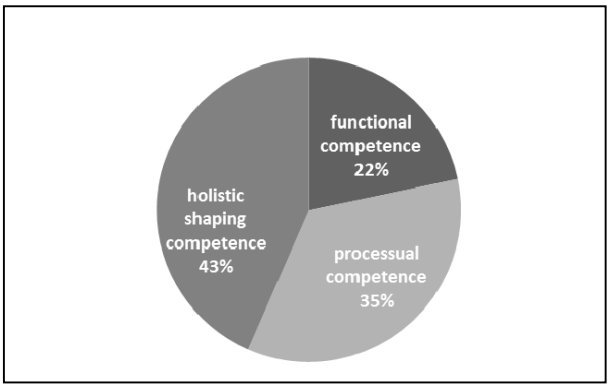


Fig. 7. Relative Share of Assessment Items per Competence Level

The results of the assessment indicate that our didactical approach worked very well with respect to problem awareness and functionality, i.e. we achieved the majority of our "technical" intended learning outcomes quite well. The "softer" intended learning outcomes have also been met, yet the improvement is less pronounced. This means that presentation, context sensitivity, and personal competencies also benefitted from our didactical approach. We suspect that the improvement can even be larger with respect to these competencies if students receive more explicit feedback.

V. Summary & Outlook

To summarize, this paper describes an inductive didactical approach to activate students in learning software engineering. In particular, the described approach focusses on understanding the need of software engineering, software process models, requirements analysis, and modelling along with respective methods and techniques. At the end of the course, students are expected to be able to practically apply their knowledge, not only to enumerate and define theoretical technical knowledge. Of course, such a participant oriented didactical approach should support further software engineering topics such as software architecture and software testing as well.

This approach is applied to students which have first contact with software engineering issues. It paves the way to train students' context-sensitive non-technical as well as technical skills in software engineering. Because this course gives students the first contact with software engineering topics these issues stay on a somewhat theoretical level in this course. To experience a "real" software engineering process in practice and hands-on, students can participate in a capstone project in the final year of their studies. In this project they can apply their theoretical knowledge in practice and be part of a software engineering team [3].

TABLE I. SECAT Evaluation Results

	Functional competence		**Processual competence**	**Holistic shaping competence**	
	Functionality	***Presentation***	***Problem awareness***	***Context sensitivity***	***Personal competencies***
Student 1	2,25	3,00	3,00	2,00	2,22
Student 2	3,25	2,00	3,25	3,00	1,33
Student 3	3,25	2,00	3,63	2,00	2,44
Student 4	2,75	3,00	2,00	3,00	2,38
Student 5	2,50	2,00	2,50	2,00	1,78
Student 6	2,75	2,00	3,25	2,00	2,44
Student 7	3,00	3,00	2,63	2,00	2,33
Student 8	3,00	3,00	3,63	2,00	2,89
Student 9	2,50	2,00	3,00	1,00	2,22
Student 10	3,25	2,00	3,63	1,00	2,44
Student 11	3,50	3,00	3,88	4,00	3,44
Student 12	3,00	3,00	3,38	3,00	3,00
Student 13	3,25	2,00	3,13	4,00	2,67
Student 14	3,00	3,00	3,43	3,00	2,56
Student 15	4,00	4,00	3,75	4,00	3,33
Mean value	3,02	2,60	3,20	2,53	2,50

ACKNOWLEDGMENT

The research project EVELIN is funded by the German Ministry of Education and Research (Bundesministerium für Bildung und Forschung) under grant no. 01PL12022A.

REFERENCES

[1] M. J. Prince and R. M. Felder, "Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases," *Journal of Engineering Education*, vol. 95, no. 2, pp. 123–138, 2006.

[2] Y. Sedelmaier and D. Landes, "Software Engineering Body of Skills," in Global Engineering Education Conference (EDUCON): IEEE, 2014, pp. 395–401.

[3] Y. Sedelmaier and D. Landes, "Practicing Soft Skills in Software Engineering," in *Overcoming Challenges in Software Engineering Education*, L. Yu, Ed.: IGI Global, 2014, pp. 161–179.

[4] D. E. Hunt, "Lehreranpassung: 'Reading' und 'Flexing'," in Berichte, Materialien, Planungshilfen / Pädagogische Arbeitsstelle, Deutscher Volkshochschul-Verband, *Sensibilisierung für Lehrverhalten: Reaktionen auf D.E. Hunts \"Teachers' adaption - 'reading' and 'flexing' to students\"*, A. Claude, Ed, Frankfurt (Main): Pädagogische Arbeitsstelle des Deutschen Volkshochschul-Verbandes, 1986, pp. 9–18.

[5] C. R. Rogers, *Freedom to learn:* A view of what education might become. Columbus, Ohio: C.E. Merrill Pub. Co, 1969.

[6] E. von Glasersfeld, "Radical constructivism and teaching," in Prospects: quarterly review of comparative education, XXXI, 2, *Constructivism and education: No 118*, C. Braslavsky, Ed.: UNESCO International Bureau of Education, 2001, pp. 161–174.

[7] P. Bourque and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge Version 3.0 - SWEBOK.* Available: http://www.computer.org/ieeecs-swebokdelivery-portlet/swebok/SWEBOKv3.pdf?token=RkjKSKMu0hA2hyGQxx5lHHB1LTkirNPu (2014, Mar. 13).

[8] Software Engineering Institute, *CMMI for Development, Version 1.3.* Available: http://www.sei.cmu.edu/reports/10tr033.pdf (2014, Oct. 24).

[9] A. Schatten, *Best Practice Software-Engineering:* Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen. Heidelberg: Spektrum Akademischer Verlag, 2010.

[10] P. Hruschka, C. Rupp, and G. Starke, *Agility kompakt:* Tipps für erfolgreiche Systementwicklung, 2nd ed. Heidelberg: Spektrum Akademischer Verlag, 2009.

[11] J. Grzega, *Learning By Teaching:* The Didactic Model LdL in University Classes. Available: http://www.joachim-grzega.de/ldl-engl.pdf (2013, Aug. 23).

[12] J. Grzega and M. Schoner, "The didactic model LdL (Lernen durch Lehren) as a way of preparing students for communication in a knowledge society," *J. of Educ. for Teaching*, vol. 34, no. 3, pp. 167–175, http://www.joachim-grzega.de/GrzegaSchoener-LdL.pdf, 2008.

[13] H. S. Barrows and R. M. Tamblyn, *Problem-based learning:* An approach to medical education. New York: Springer, 1980.

[14] M. Savin-Baden, *Problem-based learning in higher education:* Untold stories. Buckingham: Society for Research into Higher Education & Open University Press, 2001.

[15] C. H. Crouch and E. Mazur, "Peer Instruction: Ten years of experience and results," *Am. J. Phys,* vol. 69, no. 9, pp. 970–977, http://web.mit.edu/jbelcher/www/TEALref/Crouch_Mazur.pdf, 2001.

[16] H. W. Heymann, "Üben und Wiederholen - neu betrachtet," *Pädagogik (Weinheim)*, vol. 50, no. 10, pp. 7–11, 1998.

[17] H. Ebbinghaus, *Über das Gedächtnis,* 1st ed. Leipzig: Duncker & Humblot, 1885.

[18] U. Holm, *Teilnehmerorientierung als didaktisches Prinzip der Erwachsenenbildung - aktuelle Bedeutungsfacetten.* Available: http://www.die-bonn.de/doks/2012-teilnehmerorientierung-01.pdf (2014, May. 31).

[19] H. Siebert, *Didaktisches Handeln in der Erwachsenenbildung:* Didaktik aus konstruktivistischer Sicht, 2nd ed. Neuwied, Kriftel, Berlin: Luchterhand, 1997.

[20] Y. Sedelmaier and D. Landes, "A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies: SECAT - A Software Engineering Competency Assessment Tool," in 44th Frontiers in Education (FIE), 2014, pp. 2065–2072.

Anhang 1.5

Sedelmaier, Y. & Landes, D. (2015). Towards a Better Understanding of Learning Mechanisms in Information Systems Education. A Competence-Oriented Approach to Subject-Matter Didactics. In IEEE (Hrsg.), Global Engineering Education Conference (EDUCON) (S. 418–427). IEEE.

Impact-Factor (falls vorhanden): --

Eigenanteil:

Hauptautorin;

Konzeption des beschriebenen Ansatzes: Überwiegender Anteil (ca. 60 %)

Theoretische (pädagogische) Begründung des Ansatzes: Allein-Autorenschaft (100%)

Schreiben, Strukturierung und inhaltliche Überarbeitung: Gleicher Anteil (ca. 50 %)

Best-paper-award

Towards a Better Understanding of Learning Mechanisms in Information Systems Education

A Competence-Oriented Approach to Subject-Matter Didactics

Yvonne Sedelmaier, Dieter Landes
Faculty of Electrical Engineering and Informatics
University of Applied Sciences and Arts
96450 Coburg, Germany
yvonne.sedelmaier@hs-coburg.de, dieter.landes@hs-coburg.de

***Abstract*—Instructors not only in higher education are regularly faced with the problem that they need to develop a new course, or to adapt an existing one to changed requirements. This is especially true for topics related to information technology (IT) since technological progress is fast in this domain. However, instructors are not prepared really well for this task since they typically have a professional and educational background in areas different from pedagogy. Therefore, some sort of methodological framework to support the systematic development and refinement of courses would be highly appreciated.**

This paper presents such a method, called Competence-Oriented Didactics. This approach builds upon several concepts from general didactics, most notably Klafki's Didactic Analysis, and combines and extends these concepts. As a proof of concept, the method is applied to the refinement of an introductory course on information systems. This case study shows, among other things, that Competence-Oriented Didactics has the potential to be applicable for course (re-)design in other domains beyond IT as well.

Keywords—Systematic course design; Competence-Oriented Didactic; Didactc Analysis; Software Engineering Education; General Didactics; Pedagogy;

I. INTRODUCTION

Instructors in higher education are regularly faced with the problem that they need to develop a new course, or to adapt an existing one to changed requirements. This is especially true for topics related to information technology (IT) since technological progress is fast in this domain. Information technology also developed into a cross-cutting discipline that affects many other areas. Therefore, IT-related topics need to be integrated into curricula of non-IT study programs as well. Unfortunately, students in such study programs are often not aware of the role of IT and, consequently, only marginally interested.

These constraints make it even harder for an instructor to figure out which contents should be addressed in a course, how they should be arranged, and which didactical approach might be appropriate to convey the contents to the audience in such a way that understanding is maximized. This is even more difficult for instructors who are experts in their own domain, e.g. IT, but do not have a sound background in pedagogy or didactics themselves. Often, instructors are left to their own devices and approach course planning in an ad-hoc and often only loosely systematic manner. Many didactical decisions are implicit rather than conscious choices.

This contribution presents a novel approach to a more systematic course design, based on several concepts from general didactics, such as Klafki's Didactic Analysis. This approach is supposed to work for various types of courses in diverse domains. As a proof of concept, we successfully used this approach to redesign a course on information systems in a master program in financial management.

Before we present details on this case study on course redesign, we outline the motivation, theoretical underpinning and core elements of our approach. A short summary and an outlook to further refinements of our approach will conclude the paper.

II. THEORETICAL FRAMEWORKS

A. Benefit of Didactical Approaches for (Re)Designing a Course

Subject-matter didactics aim at establishing environments that promote learning. A prerequisite for any such subject-matter didactics is a better understanding of learning mechanisms. A clarification of the previous knowledge and mental concepts that students bring in a course at its outset is the basis to adapt the course to students' real needs. In case of a mismatch between students' requirements and the didactical concept, learning remains on a very simple level and will not foster relevant competencies. Course design encompasses various issues which have to be decided upon and need to be coordinated, such as the following questions:

- Which contents should be learned?
- How can instructors foster learning?
- Which methods support students' learning?

Didactics support these decisions and help creating good learning environments by systematically analyzing the specific learning situation and putting decisions on a sound theoretical basis.

B. Improving Course Designs by Didactic Analysis and Supplementary Approaches

The research reported here is part of EVELIN (Experimental Improvement of Learning Software Engineering), a project that aims at improving software engineering education through a comprehensive research design [1]. In this context, suitable didactical approaches are adapted to the specific needs of software engineering education [2], [3], [4], [5], [6], [7] in order to develop a subject-matter didactics for this domain.

In Germany, pedagogical theories have a long-standing tradition of several hundred years and are heavily influenced by events at the time of their development. Pedagogical theories mirror the specific problems and challenges of the society at their date of origin. Didactical theories make general and universally valid statements about learning, learning processes, and which determinants influence these processes. As a consequence, pedagogical and didactical theories must be put into practice and be adapted to the actual needs and requirements of the particular situation to which they are applied. In general, it is not possible to take a pedagogical framework off the shelf and use it without further refinement or adaptation. General pedagogical and didactical theories leave ample scope for interpretation which has to be filled.

Klafki's Didactic Analysis [8, 10, 9] is a well-known approach in general didactics. In this paper, this approach is utilized, adapted, and supplemented by other wide-spread didactical concepts, e.g. [11], in order to put didactical decisions on a sound basis and to develop a learning environment which satisfies students' needs. Furthermore, various other well-known didactical approaches from general didactics can be used as theoretical frameworks for (re-) designing courses in higher education. In this paper, we show how they can be applied to everyday education and combine it with elements from Didactic Analysis.

The resulting approach can be viewed as an application of general didactics as well as an extension and operationalization of Didactic Analysis that is targeting various courses in informatics on bachelor and master level, and for informatics majors as well as minors.

C. Klafki's Bildung-Centered Approach to Didactics

Each theory in general didactics presupposes a theoretical basis that helps to adapt it to specific learning situations.

1) Klafki's Theory of Education

Like many other pedagogical theories, Klafki's theory of education has a fairly philosophical and abstract underpinning. Klafki [10] is a representative of human science pedagogy (Geisteswissenschaftliche Pädagogik) and later of critical constructive theory of education (kritisch-konstruktive Didaktik). Klafki systematically analyzed contents and goals of teaching and learning processes. He viewed didactics as an instrument to clarify contents and aims of teaching. Thus, he prioritized didactics over methodological questions (primacy of didactics). This means that methodological aspects follow didactics because they depend on goal-oriented didactical decisions. Methods without didactical considerations are not helpful.

The concept of Categorical Education (Kategoriale Bildung) is at the core of Klafki's Bildung-Centered Approach to Didactics which merges "formal education" and "material education". The former focusses on methods of learning and thinking and on personal soft skills, while the latter covers a comprehensive range of topics.

Education should foster categorical education issues in both aspects. One core element is "Exemplar Approach" in the sense of Bruner's concept of "learning by discovery" [12] which means that pupils should learn things by examples which they can transfer to a more general point of view. The programming language ALGOL 60, for instance, might be taught with the expectation that students are enabled to familiarize themselves with future procedural programming languages on their own, simply because they understand the ALGOL's basic concepts and can transfer them to, e.g., the programming language C. In this context, Klafki's approach helps to take decisions on the content of education as well as on the educational substance of content. Contents which are taught should exhibit current and future significance for students.

Klafki offers Didactic Analysis as an approach to decide on the content of education (Bildungsinhalt) on the one hand, and on the substance of content (Bildungsgehalt) on the other hand.

2) Klafki's Didactic Analysis

Didactic Analysis is a pedagogical instrument which is used in teacher education for about four decades.

"The core questions of the *didactic analysis* refer to the (2) *current* and the (3) *future significance of the chosen content for the students,* which means that teachers should reflect on *educational substance* of the mandatory curriculum. Question 1 focuses on the aforementioned *exemplarity of the chosen topic*, question 4 on its *structuredness*, and question 5 on the *accessibility* of the topic (=methodical aspects of teaching)." [13]

This quote refers to the five core questions in Klafki's Didactic Analysis:

"I. What wider or general sense or reality do these contents exemplify and open up to the learner? What basic phenomenon or fundamental principle, what law, criterion, problem, method, technique, or attitude can be grasped by dealing with these contents as 'examples?'

[...]

II. What significance does the content in question, or the experience, knowledge, ability or skill to be acquired through this topic already possess in the minds of the children in my class? What significance should it have from a pedagogical point of view?

[...]

III. What constitutes the topic's significance for the children's future?

[...]

IV. How is the content structured (which has been placed in a specifically pedagogical perspective by questions I, II and III)?

[...]

V. What are the special cases, phenomena, situations, experiments, persons, elements of aesthetic experience, and so forth, in terms of which the structure of the content in question can become interesting, stimulating, approachable, conceivable, or vivid for children of the stage of development of this class." [9]

These questions are related and intertwined with each other to provide a complete picture of a course.

3) Evaluation of Klafki's Approach

Klafki's Didactic Analysis is a methodological framework that is intended to support the systematic analysis of courses. The resulting prioritization of contents allows focusing on particular issues in such a way that the covered topics become more easily understandable for the intended audience.

Initially, Didactic Analysis neglected methodological aspects completely.

Furthermore, Klafki's general framework needs to be supplemented by additional didactical approaches. While Klafki's approach clarifies the educational content in general, it lacks support for choosing specific methods or media. Even though Klafki extended Didactic Analysis in this respect [14], it still falls short of being an applicable framework for planning single lessons. In order to close the remaining gap, we combined Didactic Analysis with elements of other didactical approaches such as [11]. Consequently, the resulting framework is a tool to decide on relevant content for a particular audience, but also on which ways are best suited for teaching and learning these contents.

Moreover, Klafki's approach considers learning conditions and influencing factors only implicitly.

Yet, the primacy of didactics from Klafki's theory can be generalized. It does not seem to be helpful to choose some methods or media without knowing the intended learning outcomes.

Furthermore, thinking about the educational substance and the significance of the chosen content seems to be worthwhile for teachers and lecturers.

D. The Learning-Centered Approach to General Didactics (Berlin Model)

Paul Heimann, one of Klafki's scholars, devised the Berlin Model in order to create a more practical approach for every-day use in education. Heimann's Berlin Model aims at enabling teachers to analyze their lessons on a purely empirical and objective basis. The Berlin Model was developed to make educational decisions more transparent and to assist teachers considering as many factors as possible in planning their lessons. Other than Klafki's "developmental education theory" (Bildungstheoretische Didaktik), the Berlin Model of Lesson Planning [11] does not think in purely ideal terms of human science pedagogy (Geisteswissenschaftliche Pädagogik), but wants to support practical decisions of teachers by comprising all planning-relevant factors influencing teaching and learning.

In the Learning-Centered Approach, the focus shifts from "Bildung" (Klafki) towards learning. Didactics focus no longer on personality, but rather on being part of the society. Therefore,

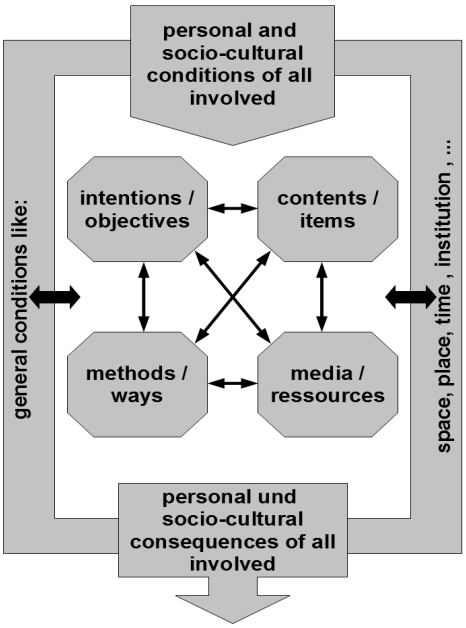


Fig. 1. Berlin Model (http://commons.wikimedia.org/wiki/File:BerlinerModellEN.jpg#mediaviewer/File:BerlinerModellEN.jpg)

cultural aspects become less important in favor of contents of learning and information. Now teaching with emphasis of effective organization of learning processes becomes increasingly relevant for didactics. This change of mind went along with a more practical view on didactics.

Heimann's scholar Schulz distinguished four fields of decision making (see fig. 1):

- Intentions;
- Themes, topics;
- Methods;
- Media.

These areas of decision are constrained by two types of conditions: socio-cultural and individual conditions of students.

These aspects offer a comprehensive view on inter-relationships of learning processes. Therefore, this approach seems to be quite appropriate to analyze and plan courses since it is general enough to analyze nearly all kinds of lessons, independent of the subject.

Yet, in contrast to most pedagogical theories, the learning-centered approach to general didactics lacks an underlying learning theory, i.e. neither offers any specific definition for learning, nor builds on specific assumptions of how learning works [15].

E. The Hamburg Model of Lesson-Planning (Hamburger Modell)

Schulz revised the Berlin Model in order to emphasize teachers' critical reasoning on the societal dimension of schooling. By then, Schulz had moved from Berlin to Hamburg. Thus, the resulting model was called Hamburg Model [16].

Schulz emphasized three main educational objectives: competence, autonomy, and solidarity. As a consequence, tuition needs experience of theme, of feel, and of social experience. These aspects reflect Ruth Cohn's Theme-Centered Interaction (TCI) by balancing "I" (individual subject), "it" (theme), and "we" (group).

Schulz established five steps for lesson planning [17]:

- Criteria of planning with focus on teaching, e.g. action-orientation;
- Four areas of decision making and two areas of conditions according to the Berlin Model;
- Didactical activities, such as advising, evaluating, analyzing, planning, realizing, and executing and acting cooperatively;
- Fundamental thoughts on lesson planning. To achieve the aforementioned three main goals of education, namely competence, autonomy, and solidarity, learners have to experience themes in three aspects: things, feelings, and social experience;
- Levels of planning.

The latter relies on different aims of lesson planning, namely prospective planning, outline planning, process planning, and planning corrections. Prospective planning has an extended time perspective, e.g. a term, a calendar year, or a school year. Outline planning focuses on individual teaching units. Process planning arranges topics in time. Planning corrections allow teachers to react on unexpected situations during the course.

All in all, the Hamburg Model expands the Berlin Model's decision and condition areas with societal issues. This introduces a normative component into the theory, an aspect which was widely criticized.

Furthermore, a more theoretical framework is built around these fields by defining the aforementioned five steps of lesson planning. These steps add new aspects and questions which should be considered when thinking about didactics.

F. Critical Constructive Approach to Didactics

Under the impression of the German student protest movement in the late 1960s, Klafki elaborated his model to the Perspective Scheme of Lesson Planning. His concept of education renewed the idea of general education by relating it to the Critical Theory of Society of the Frankfurt School (Habermas, Horkheimer). Klafki focusses on emancipation as a central goal of education. He also defined Key Problems of the Modern world [18] such as peace, environment, unequal distribution of wealth, (un-)employment, relationship between generations and between men and women. Educational contents should reflect these key problems.

New concepts of Didactic Analysis include methodological aspects [14] and, in his later work [9], even Klafki highlights the connection between intention-oriented and methodological aspects of planning a course.

In his new approach, learning becomes more interactive instead of only focusing on contents.

Still, Klafki's approach remains quite static because it focuses on analyzing structures after all. Dynamic elements such as process planning remain implicit only.

III. A Novel Approach of Planning, Structuring and Initiating Learning Processes (Competence-oriented Didactics)

Klafki's approach for analyzing and planning a curriculum is based on five key questions. Our approach takes Klafki's Didactic Analysis as a starting point and adapts it to our specific needs, e.g., for analyzing a course such as "Information and Communication Systems".

We extend and combine Klafki's approach of Didactic Analysis with elements from other educational theories (see sec. II).

This results in a so-called Competence-Oriented Didactics (see fig. 2) which contains elements from the aforementioned didactical theories supplemented by summative and formative evaluation [19] issues.

As a starting point, Klafki's Didactic Analysis clarifies the educational content and substance. Competence-Oriented Didactics follow Klafki's primacy of didactics. This means that the educational objectives are defined at the beginning of the planning process and serve as compass for the further planning steps.

The Berlin Model's two condition areas are added to Competence-Oriented Didactics to analyze the constraints of the planned learning processes.

The following planning levels, namely prospective, outline, and process planning are inspired by the corresponding elements of the Hamburg Model. The four decision areas of the Berlin Model are merged into these planning steps. Prospective planning focuses on content and structural aspects which are specified in outline planning which also introduces methodological aspects. Process planning emphasizes methods and media which are used in one specific lesson. The level of detail concerning methods and media increases in parallel with the level of detail in operationalizing educational objectives.

After planning a learning situation from general to detail, realization takes place. Within this phase, planning correction as mentioned in the Hamburg Model is necessary. In addition, elements from constructivist didactics are used, e.g. participant orientation [20]. Hunt describes this as "reading" and „flexing" [21]: Reading means attentively observing students, while flexing concerns reacting on recognized requirements and needs. This generates an iterative process of adapting teaching and learning.

In order to improve learning processes the phase "evaluation" completes Competence-Oriented Didactics. The aforementioned didactical theories neglect evaluation in unison.

Evaluation in Competence-Oriented Didactics operates on two levels, namely summative and formative evaluation [19]. In this didactical concept, summative evaluation is concerned with the assessment of students' competencies by trying to find out, which competencies they developed, for example by writing an exam. Formative evaluation focuses on the effectiveness of the

used didactical approach, especially the chosen methods and media. Therefore, students' competence developments as a whole are the basis to draw conclusions on the didactical approach. Software Engineering Competency Assessment Tool (SECAT) as an evaluation tool rests on a sound theoretical basis and initiates further refinement of the didactical approach in order to provide a subject-matter didactics [7].

A. *Definition of Terms and Learning Theory of Competence-Oriented Didactics*

It is assumed that learning processes should foster students' competencies. Competencies constitute the capability to cope with complex and new situations and also presuppose additional skills, which are often subdivided into social, personal, and methodological competence [23, 22, 24].

In this paper, the term "competence" is not used for factual or technical knowledge since, according to Weinert [25] (p. 35), a "skill is an ability to perform complex motor and/or cognitive acts with ease, precision, and adaptability to changing conditions". Following this view, neither soft skills, nor factual knowledge in isolation are competencies. Competencies can only come into existence when both interact: "Competence" presupposes technical or factual knowledge and also soft skills. Moreover, competence encompasses the context, emotional elements, and also possesses an ethical, normative component. Competence enables individuals to analyze complex and new situations, to find creative potential solutions, and to decide on one way of action, in due consideration of causes and consequences. Competence also includes the willingness and motivation to act autonomously and based on self-initiative after a cognitive analysis of a situation. We distinguish context-sensitive soft skills, generic soft skills, and factual knowledge [3].

University education should foster students' competencies. But neither are there clear cause-effect-relationships, nor can competency be traced back to one didactical approach or method. Nonetheless, higher education should address competencies and create environments that promote students' competencies in accordance with constructivist didactics.

B. *Key Questions of Competence-Oriented Didactics*

The core questions we used for analyzing, planning and realizing learning situations are the following (for their interrelationships see figure 2):

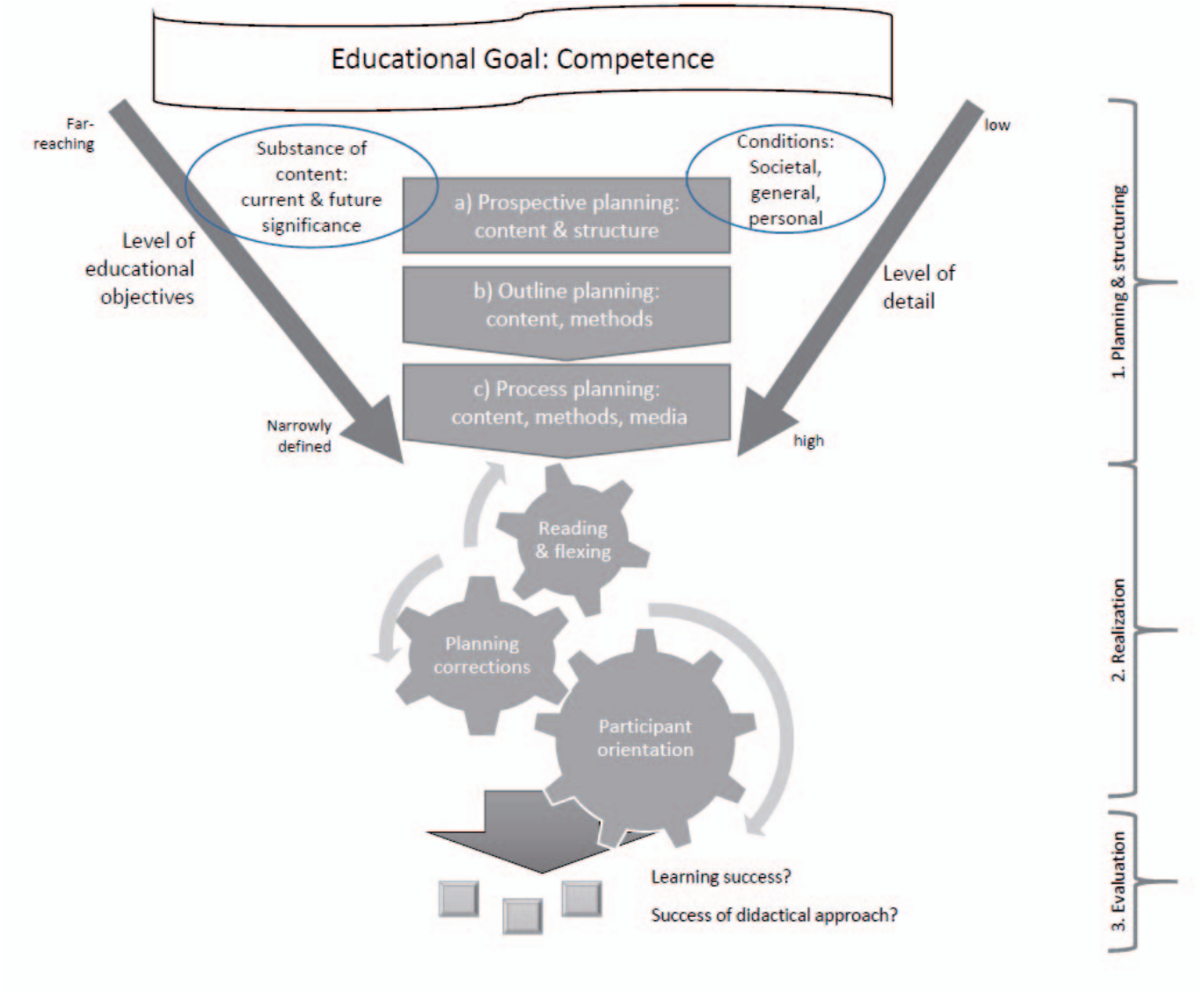


Fig. 2. Competence-Oriented Didactics

1) Planning & Structuring

- Conditions: Societal, general, personal: What are the basic conditions? Which organizational restrictions do exist for this course?
- Substance of Content: What are the main ideas and goals of the course (general goals)? (Primacy of Didactics)
- How is the topic transformed into a key question?
- Current and future significance: Why should students in a particular study program cope with a particular topic? Where in their lives did or will students have contact to these subjects? What exactly is the educational merit and substance of the issue?

a) Prospective planning

- What is a useful structure of contents?
- How does the background of students influence learning? Do students have prior knowledge? Are there wrong assumptions or misconceptions?

b) Outline planning

How is the subject accessed? How can students acquire interest for the subject? How can learning situations address objects, feelings, and social experiences? Is an inductive way of learning more appropriate than a deductive one? Is the concept based on an inductive or deductive way of learning? Does it bring students from general to detailed aspects or vice versa? Should students learn some examples which are exemplary for the whole topic (learning by example) [26] or should they deduce practical knowledge from general regularities? Should they transfer general laws to real world problems by themselves? Access to the subject is also determined by the level of activation that instructors enforce, i.e. do they want students to participate actively in the learning process or do they apply instructive ways of teaching? An additional question is whether the subject is approached from a problem or from a solution. Which didactical principles [27] are recognized and are they tailored to the goals? What about repeating contents? When and how?

c) Process Planning:

- How are the educational objectives operationalized to intended learning outcomes?
- Which methodological approach shall be taken for a specific topic under specific circumstances considering the educational objectives? Which methods and media are chosen and why?

2) Realization

Planning corrections: How and what type of information do instructors collect regularly in order to stay current with students' learning processes and to provide feedback? How do they recognize if students understand the matter, where they have misconceptions, and what they were surprised about?

3) Evaluation

How are students' competencies assessed? There should be some kind of measurement if they achieved the educational objectives and fostered their competencies.

Furthermore, there should be an evaluation of the applied didactical approach to improve learning continuously. Did the applied didactical approach work well? Why or why not?

IV. Case Study: Adapting and Applying Didactical Approaches in an Information Systems Course

As a representative case, we applied this operationalization of Competence-oriented didactics to a course "Information and Communication Systems" in a master program on Financial Management. As a baseline, we use the shape of the course as it was held in the winter term 2012/2013 and refer to changes we introduced in the winter terms of 2013/2014 and 2014/2015.

The master program on Financial Management is not IT-related, but rather focusses clearly on economic issues. An additional, specific challenge arises from the fact that it is an international study program. Thus, there are different geographical and professional cultures, and the mindsets of economists and software engineers or IT specialists have to be aligned, even more so since this is not just a requirement for a university course: Also in professional practice, the two worlds need to meet each other. Software engineers or IT specialists have to cooperate with economists: Economists express what they expect from software, and software engineers develop software by relying on this information. Unfortunately, in practice there are various challenges when computer scientists and economic persons work together.

Below, some of the challenges of teaching information systems to business administration students will be discussed in more detail, and conclusions for a subject-matter didactics will be drawn from this analysis. There will also be an outline of how didactical approaches can be used to analyze and design such a course in a goal-oriented manner, for instance by clarifying the reasons for, e.g., methodological decisions.

A. Conditions: Societal, General, Personal

The MBA program Financial Management at Coburg University presupposes a bachelor degree in business administration plus work experience in this field. The program is designed for an international audience. As a consequence, an audience of 20 to 40 people typically originates from more than a dozen different nations. Therefore, the acquaintance of such a heterogeneous audience with IT in general and information systems in particular also varies greatly: Some students worked as consultants for ERP systems over several years, while others do not know IT systems beyond typical text processing or presentation software.

Due to their different origin, also cultural differences within the group need to be taken into account. In addition, English is the least common denominator for communication within the group and with instructors although it is the native language of only a small minority of students, if any.

Besides an introduction to ERP systems, Information and Communication Systems is the only course in the curriculum with a somewhat technical focus. Information and Communication Systems has two contact hours per week over a term of 15 weeks; its workload is equivalent to five credit points. At the end of the term, there is a written examination of 45 minutes.

The general settings for the course did not change over the last few years.

B. Substance of Content

This section aims at characterizing the main ideas and goals as well as transforming the subject matter into key questions (see section III.B.1).

The key question for the course can be phrased as "Which benefits do information systems provide for me in my job after finishing the Financial Management program?". This general theme can be broken down into more specific questions, such as:

– How do information systems work?

This touches upon the required technologies (databases, communication technology), but also refers to software architectures of information systems (client/server, multi-tier).

– How are information systems used in practice?

This question leads to typical categories of information systems (B2B / B2C / Computer-supported cooperative work, Transaction processing / Management information / Decision support / Executive information systems).

– What are limits and risks in using information systems and how can I handle them?

This encompasses aspects of IT security and privacy, including hazards such as malicious software and countermeasures (security updates, antivirus software, firewalls).

– Which role do I play during the selection / development of a suitable information system?

Here, it will be pointed out that in the software lifecycle particularly requirements analysis and (acceptance) testing are relevant for non-IT persons. Furthermore, the "interfaces" between non-IT persons and software engineers are clarified, in particular with respect to the information on processes and data that must be conveyed from the application domain to the development organization. This is also meant to provide a clearer picture of the different mind sets of non-IT and IT persons.

Although the key questions did not change with respect to the situation before applying Competence-oriented Didactics, they had not been expressed so explicitly. In our baseline, a written syllabus was primarily focusing on contents rather than intention of the course.

C. Current and Future Significance

Characterizing the significance aims at identifying areas where the subject matter has been or will be relevant for the target audience (see section III.B.1). These issues touch upon why students should get involved in the subject and which benefit they can expect from doing so.

Typically, all Financial Management students have some acquaintance with text processing or presentation software, and all of them are familiar with using the internet, both for private purposes and during their bachelor education or in the workplace.

Yet, only few of them possess a deeper knowledge of the underlying technology. For instance, they know that IP addresses are important with respect to the internet, but they do not really understand why. Basic technological knowledge will help them to gain a better and more coherent picture of how different technologies fit together. Consequently, students will be better prepared to cope with unexpected things in complex situations in their everyday business.

Furthermore, students tend to use the internet somewhat airily. Deeper insights into security issues will increase their awareness of potential risks and consequences of neglecting security aspects. As a consequence, they will also act more cautiously, for instance when they can distinguish a secure URL from an unsecure one or use encrypted communication channels via HTTPS.

Finally, a better understanding of the technological potential in IT leads to a change in perspective. Originally, students tend to assume that they have to take software as it is: If there is an appropriate piece of software, it can be used; if there is none, there is nothing that can be done about it. This attitude is replaced by the insight that, especially in a professional context, software might be built to their needs. Yet, this presupposes that they are able to express their requirements in a way that can be understood by other disciplines, namely by software engineers. Cooperation with other disciplines is inevitable.

D. Prospective Planning

Prospective planning aims at clarifying which contents are relevant for the audience, given the general substance of the matter.

Prospective planning is heavily constrained by some of the context conditions, namely the professional background of the participants as well as their previous knowledge. In order to clarify these issues further it is also desirable to uncover misconceptions that participants might have with respect to some of the material to be covered in the course.

In order to gain insight into these issues, we asked participants to introduce themselves in a short presentation, highlighting their geographical origin, their professional experience (including higher education), their knowledge on IT, and their expectations for the course. In particular in the winter term 2014/2015, this clearly showed that the previous knowledge of participants with respect to IT is fairly poor, the audience is extremely heterogeneous in terms of professional experience and studies, and misconceptions with regard to the goals and contents of the course were pretty common.

As a consequence, we tried to straighten out these misconceptions by highlighting which expectations might be fulfilled in the course, and which would not.

In parallel, contents needed to be rearranged and additional, more basic issues needed to be introduced in order to cope with the low level of previous knowledge.

An important issue is finding an appropriate alignment of topics (see section III.B.1). Competence-oriented Didactics implies a different access to the subject matter based on more realistic case studies and examples. In the baseline, contents

were often presented instructively by first explaining technologies and then showing their potential usages.

This is now somewhat reversed. Primary emphasis is now put on specific goals that need to be achieved by using an information system, and then deriving what technological issues are required for meeting these goals. For instance, the point that information systems should be distributed led to the required communication technologies, e.g. TCP/IP or HTTP, and to software architectures such as client/server or multi-tier. Distributed information systems that exchange potentially confidential information over the internet then lead to an understanding of the reasons why a VPN connection might be required to safeguard information, and why it might be wise to prefer a slower, but more secure connection over of a fast, but insecure one.

E. Outline Planning

Outline planning is concerned with planning how a particular subject will be accessed, in particular in order to engage students in the subject since they find it interesting in some way (see section II.B.1).

In our baseline, contents were presented in a somewhat abstract manner which was easily intelligible for those students that were already somewhat familiar to the matter. "Newcomers", however, could hardly imagine what the presented material meant precisely. As an instance of active learning, students were required to investigate a technological aspect in teams of two or three students. Findings were reported back to their classmates in presentations of roughly 15 minutes each, and written reports of 5 to 10 pages were prepared for each subject and provided on a file server. This approach, however, fell short of generating a broader understanding – each group only understood their own topic well, but often failed to see the point in other topics.

Thus, Competence-oriented Didactics required changing that approach in order to live up to the key questions raised. Emphasis moved away from theoretical technical knowledge and now lies on the context of information systems in an economic world. The subject matter is transferred closer to students' reality, thus making it more accessible and more interesting. By and large, the main emphasis was put on creating a better understanding, e.g. of how computer networks and the internet operate, which roles particular networks play in this context, etc.

To that end, we used three case studies that showed how information systems were used in a realistic setting. Thus, technical content was linked to the students' economic background and their prior knowledge. As a side effect, this approach made information systems more familiar to students and less technical; it became clearer to students why it could be helpful to understand secure issues and types of connections.

In a similar vein, additional topics were conveyed using close-to-life examples from their world. For instance, different categories of information systems, such as transaction processing or decision support systems, were made more tangible by discussing how a company such as Amazon might use them.

Furthermore, Competence-oriented Didactics revealed that the choice and order of presentation of issues was not ideal to meet the key questions. Therefore, we slightly adapted them and put considerably less emphasis on databases. Instead, we put more stress on describing required or offered services of information systems, thus laying the basis for expressing requirements as use cases [1].

Awareness for IT security was raised with questions such as "How can I make sure my money is safe when I do internet banking?", "How can I recognize phishing mails?", or "Under which conditions should I give my banking credentials?". Again, these questions are close enough to students' reality to raise interest.

As it turned out, this approach of transferring the technical content to scenarios that are familiar to the students worked well in stimulating lively discussions in which almost everybody in the course got involved.

F. Process Planning

Process planning aims at identifying the didactical methods that are appropriate to achieve teaching and learning goals for a course. In general, this includes the formulation of tasks that students need to work on, but also considerations of how a link to the topics discussed in the last class could be established.

It was already mentioned above that contents were often presented instructively in the baseline. Additionally, distinct technological topics had to be investigated and presented in a group assignment.

Competence-oriented Didactics indicated that this approach was not completely convincing since students had a tendency to remain passive instead of getting actively involved.

Therefore, the methodology of teaching changed in the current term. Now, the instructor leads students through a process of building up a comprehensive picture of internet. The structure of the internet constitutes the starting point for this process; afterwards information is added on how communication over the internet works, and then the transport of data through the internet is explained. Thus, technological challenges and obstacles became obvious for students. As a consequence, they developed an awareness and understanding for security aspects and how they work. All this was developed in a joint effort in lively discussions which were guided by the instructor and involved almost all of the students. Now, they developed interest and asked questions by themselves without being expressly being urged to do so.

The changes in emphasis and structure of contents combined with new real-world case studies led to other methodological decisions: the students learned a smaller share of material by hearing theoretical knowledge as before in lectures, but an increasing portion by working together in small groups and plenary discussions.

As one characteristic example, occupation with security aspects was also turned from a group assignment into a discussion. Why did we do that? In the baseline, a group work based on problem-based learning was employed to give students an understanding of security aspects when using the internet. Even though they were provided with hints to literature, it turned

out to be quite difficult for them to summarize the relevant information and present it to their peer students.

Currently, group assignments were still used, but their character and mission changed significantly. As an initial group assignment, students have to work on a case study in small groups of approximately 7 students. In this group work, students need to solve a given problem in the context of the case study, reflect on possible solutions, and finally make a proposal for a possible solution on a flipchart or an overhead slide. This was different to the baseline course where we tended to start out from a solution, but students occasionally did not even see the problem that was to be solved. Due to Competence-oriented Didactics, the methodology was inverted and now starts out from a problem that has to be solved; only in a second step a solution is worked out. This approach can be allocated to problem-based learning [28] and was used once at the beginning of the term.

The initial case study was referenced as a running example during the whole term. So, we achieved a common example in a heterogeneous group of students. This case study also initiated certain group dynamics. Students adopted an active behavior over the whole term because they were encouraged to do so from the first class on. By applying more learner-activating methods the course became livelier and students took part actively. They asked more and frequently also unsolicited questions and were encouraged to think about the theme. This also led to more interaction within the class and to technical discussions among the students.

Competence-oriented Didactics also showed that the heterogeneity of the audience was only taken into account insufficiently in the baseline course. Problem-based group assignments also address this issue. Since these assignments are treated in small peer groups, students take courage to ask also seemingly stupid questions to their peers. If there are knowledgeable colleagues in the group, the question can be answered on the spot; if not, the question apparently is not so stupid, and therefore one does not make a fool of oneself if the instructor is asked for help. In a certain sense, this brings in an additional aspect that can be allocated to peer instruction since there is a direct transfer of knowledge among students.

In summary, more learner-activating methods for the student group as a whole were applied continuously over the complete term instead of using problem-based learning approaches in smaller groups. The latter was employed only once at the beginning of the term.

G. Realization

An important aspect in realizing the didactical concept is the flexible reaction to difficulties that students encounter. As a prerequisite, instructors need to get aware of difficulties and misconceptions. This can be accomplished by asking questions in order to get impression of the degree of understanding from the given answers, and observing and coaching students while they were working on a task assignment in class.

In the particular course, it became apparent that students did not get a clear picture of why network protocols are needed in computer networks. We tackled this by addressing this point again, using a different and more accessible scenario.

In the last course, we made the additional observations that students stayed quite passive and hardly took part in the lessons actively. To involve students and increase their motivation to play a more active role in the lesson we pursued two main approaches:

First, students were given clear and precisely written tasks at the beginning of group assignments. This should give them more confidence in doing the right thing by giving them a better idea of instructors' expectations.

Second, group work was split into a two-step process: In the first step students assemble in small groups of 5 persons to discuss a specific topic. Then, in a second step, instructors brought together 2 to 3 groups which had worked on the same tasks to merge their results to one result. This yields several good results because none of the groups developed a complete solution alone, but rather brought in just some part of an overall solution while neglecting some aspects that other groups brought in.

V. Summary And Outlook

The application of didactical knowledge leads to a better understanding of learning and allows instructors to match specific learning environment and methods with students' requirements. The paper provides evidence which was gained from applying didactical approaches as a scientific discipline [29]. It describes the changes and improvements which are caused and triggered by applying didactical concepts systematically.

In particular, we used well-known elements from general didactics and combined and extended them, giving rise to a novel approach called Competence-oriented Didactics. The latter constitutes a scientific approach to plan and design a course in engineering education. In particular, this approach builds on Klafki's Didactic Analysis. Yet, Klafki's Didactic Analysis cannot be used right out of the box, but rather needs to be operationalized to be usable in practice, also taking influence from other models, such as the Berlin and Hamburg Model, into account.

The resulting approach is applied successfully to a representative example, namely a course on information and communication systems in an MBA program. This shows how approaches from general didactics as a discipline in its own right support analyzing, improving, and further developing higher education in computer sciences. It helps instructors to ground their didactical decisions on a solid pedagogical underpinning and shows a way how instructors can benefit from thinking about didactics.

We applied Competence-Oriented Didactics to several courses in higher education of informatics. Although small adaptions might be required for other domains, Competence-Oriented Didactics seems to be general and beneficial enough to be widely applicable.

Acknowledgment

The research project EVELIN is funded by the German Ministry of Education and Research (Bundesministerium für Bildung und Forschung) under grant no. 01PL12022A.

REFERENCES

[1] Y. Sedelmaier and D. Landes, “A Research Agenda for Identifying and Developing Required Competencies in Software Engineering,” *International Journal of Engineering Pedagogy (iJEP)*, vol. 3, no. 2, pp. 30–35, 2013.

[2] Y. Sedelmaier and D. Landes, “A Multi-Level Didactical Approach to Build up Competencies in Requirements Engineering,” in 8th International Workshop on Requirements Engineering Education & Training (REET 2014): CEUR Workshop Proceedings, 2014, pp. 26–34.

[3] Y. Sedelmaier and D. Landes, “Software Engineering Body of Skills,” in Global Engineering Education Conference (EDUCON): IEEE, 2014, pp. 395–401.

[4] Y. Sedelmaier and D. Landes, “Using Business Process Models to Foster Competencies in Requirements Engineering,” in 27th International Conference on Software Engineering Education and Training (CSEE&T), 2014, pp. 13–22.

[5] Y. Sedelmaier and D. Landes, “Practicing Soft Skills in Software Engineering,” in *Overcoming Challenges in Software Engineering Education*, L. Yu, Ed.: IGI Global, 2014, pp. 161–179.

[6] D. Landes, Y. Sedelmaier, V. Pfeiffer, J. Mottok, and G. Hagel, “Learning and teaching software process models,” in Global Engineering Education Conference (EDUCON), 2012, pp. 1153–1160.

[7] Y. Sedelmaier and D. Landes, “A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies: SECAT - A Software Engineering Competency Assessment Tool,” in 44th Frontiers in Education (FIE), 2014, pp. 2065–2072.

[8] W. Klafki, “Didaktische Analyse als Kern der Unterrichtsvorbereitung,” in Auswahl, Grundlegende Aufsätze aus der Zeitschrift Die deutsche Schule Reihe A, vol. 1, *Didaktische Analyse*, H. Roth and Blumenthal Alfred, Eds, Hannover: Schroedel, 1964, pp. 5–34.

[9] W. Klafki, “Didactic analysis as the core of preparation of instruction (Didaktische Analyse als Kern der Unterrichtsvorbereitung),” *Journal of Curriculum Studies*, vol. 27, no. 1, pp. 13–30, 1995.

[10] W. Klafki, *Studien zur Bildungstheorie und Didaktik*. Weinheim: Beltz, 1965.

[11] P. Heimann, G. Otto, and W. Schulz, Eds, *Unterricht:* Analyse und Planung. Hannover: Schroedel, 1965.

[12] J. S. Bruner, *On knowing:* Essays for the left hand, 5th ed. Cambridge Massachusetts, 1969.

[13] K.-H. Arnold, “Didactics, Didactic Models and Learning,” in *Encyclopedia of the Sciences of Learning*, N. M. Seel, Ed.: Springer US, 2012, pp. 986–990.

[14] W. Klafki, *Neue Studien zur Bildungstheorie und Didaktik:* Beiträge zur kritisch-konstruktiven Didaktik. Weinheim: Beltz, 1985.

[15] W. Jank and H. Meyer, *Didaktische Modelle,* 7th ed. Berlin: Cornelsen Scriptor, 2005.

[16] W. Schulz, “Ein Hamburger Modell in der Unterrichtsplanung: Seine Funktionen in der Alltagspraxis,” in *Didaktische Modelle und Unterrichtsplanung*, B. Adl-Amini and R. Künzli, Eds, München: Juventa, 1980, pp. 49–87.

[17] F. W. Kron, *Grundwissen Didaktik,* 5th ed. München, Basel: E. Reinhardt, op. 2008.

[18] W. Klafki, *Neue Studien zur Bildungstheorie und Didaktik:* Zeitgemäße Allgemeinbildung und kritisch-konstruktive Didaktik, 4th ed. Weinheim: Beltz, 1994.

[19] J. Reischmann, *Weiterbildungs-Evaluation:* Lernerfolge messbar machen. Neuwied: Luchterhand, 2003.

[20] U. Holm, *Teilnehmerorientierung als didaktisches Prinzip der Erwachsenenbildung - aktuelle Bedeutungsfacetten.* Available: http://www.die-bonn.de/doks/2012-teilnehmerorientierung-01.pdf (2014, May. 31).

[21] D. E. Hunt, “Lehreranpassung: 'Reading' und 'Flexing',” in Berichte, Materialien, Planungshilfen / Pädagogische Arbeitsstelle, Deutscher Volkshochschul-Verband, *Sensibilisierung für Lehrverhalten: Reaktionen auf D.E. Hunts \"Teachers' adaption - 'reading' and 'flexing' to students\"*, A. Claude, Ed, Frankfurt (Main): Pädagogische Arbeitsstelle des Deutschen Volkshochschul-Verbandes, 1986, pp. 9–18.

[22] L. Reetz, “Zum Zusammenhang von Schlüsselqualifikationen - Kompetenzen - Bildung,” *Aus Politik und Zeitgeschichte. Beilage zur Wochenzeitung Das Parlament*, no. 37, pp. 13–20, http://www.sowi-online.de/reader/berufsorientierung/reetz_lothar_1999_zum_zusammenhang_von_schluesselqualifikationen_kompetenzen_bildung.html, 1999.

[23] H. Orth, *Schlüsselqualifikationen an deutschen Hochschulen:* Konzepte, Standpunkte und Perspektiven. Bielefeld: UVW, Webler, 1999.

[24] D. Schneckenberg, *Educating Tomorrow's Knowledge Workers: The Concept of ECompetence and Its Application in International Higher Education*: Eburon, 2008.

[25] F. E. Weinert, “Concept of competence: A conceptual clarification,” in *Defining and selecting key competencies*, D. S. Rychen and L. H. Salganik, Eds, Seattle: Hogrefe & Huber, 2001, pp. 45–65.

[26] M. Wagenschein, *Verstehen lehren*. Weinheim: Beltz, 1968.

[27] H. Siebert, *Didaktisches Handeln in der Erwachsenenbildung:* Didaktik aus konstruktivistischer Sicht, 2nd ed. Neuwied, Kriftel, Berlin: Luchterhand, 1997.

[28] J. R. Savery, “Overview of Problem-based Learning: Definitions and Distinctions,” *Interdisciplinary Journal of Problem-based Learning*, vol. 1, no. 1, 2006.

[29] P. Figas, S. Knörl, S. Mörtlbauer, Y. Sedelmaier, and I. Schroll-Decker, “Software Engineering Education as a Didactical Discipline in its own right,” in 1st European Conference on Software Engineering Education (ECSEE), 2014, pp. 1–15.

Anhang 1.6

Sedelmaier, Y. & Landes, D. (2015). Überfachliche Kompetenz im Software Engineering - Modellierung, Förderung und Messung in der Hochschulausbildung. In U. Riegel, S. Schubert, G. Siebert-Ott & K. Macha (Hrsg.), Kompetenzmodellierung und -messung in den Fachdidaktiken (S. 111–130). Münster: Waxmann.

Impact-Factor (falls vorhanden): --

Eigenanteil:

Alleinautorin;

Konzeption des beschriebenen Ansatzes:	Allein-Autorenschaft (100%)
Theoretische (pädagogische) Begründung des Ansatzes:	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung:	Allein-Autorenschaft (100%)

Sonderdruck aus

Ulrich Riegel, Sigrid Schubert,
Gesa Siebert-Ott, Klaas Macha (Hrsg.)

Kompetenzmodellierung und Kompetenzmessung in den Fachdidaktiken

ISBN 978-3-8309-3236-9

Steinfurter Str. 555, 48159 Münster

Bestellung per Fax: 0251 26504-26 oder
telefonisch: 0251 26504-0;
per Internet unter www.waxmann.com/buch3236 oder
per E-Mail: order@waxmann.com

Überfachliche Kompetenzen im Software Engineering
Modellierung, Förderung und Messung in der Hochschulausbildung

Yvonne Sedelmaier & Dieter Landes

Software ist zentraler Teil unseres Alltags und muss mit Hilfe ingenieurgemäßer Vorgehensweisen, dem sogenannten Software Engineering, entwickelt werden. Software Engineering ist aufgrund der Komplexität von Software-Entwicklungs-Projekten sowie der Vielzahl an erforderlichen Kompetenzen nur schwierig in der Hochschulausbildung abbildbar. Neben fachlichen Kompetenzen rücken zunehmend auch überfachliche Kompetenzen in den Fokus der Ausbildung. Dieser Beitrag beschäftigt sich vorrangig mit denjenigen überfachlichen Kompetenzen, die einen Software-Ingenieur charakterisieren und ihn von anderen Fachwissenschaften abgrenzen. Dazu wird eine Unterscheidung in kontextsensitive überfachliche sowie allgemeine überfachliche Kompetenzen getroffen, die zusätzlich zum Fachwissen eines Informatikers erforderlich sind. In diesem Beitrag wird eine Forschungsmethodik dargestellt, wie mit einem qualitativen Forschungsansatz, der auf der Grounded Theory (Glaser & Strauss 2010) basiert, diese kontextsensitiven überfachlichen Kompetenzen erhoben und beschrieben wurden. Ferner wird ein Beschreibungsmodell erläutert, das diese Kompetenzen sowie deren Zusammenhänge angemessen abbilden kann. Der entstandene „Software Engineering Body of Skills" (SWEBOS) (Sedelmaier& Landes 2014c) beschreibt die erforderlichen kontextsensitiven überfachlichen Kompetenzen eines Software-Ingenieurs und setzt somit eine Zielmarke für die Hochschulausbildung im Software Engineering.

1 Begriffe und Einordnung

1.1 Software Engineering

Software beeinflusst mittlerweile viele Bereiche unseres Alltags. Viele Geräte werden mittels Software gesteuert, angefangen vom Herzschrittmacher, über Waschmaschinen und Airbags in Autos bis hin zu Mobiltelefonen (Sommerville 2011). Darüber hinaus übernehmen neue Medien und mobile Geräte wie das Internet und Smartphones immer mehr Funktionen in unserem Leben. Wir buchen Fahrkarten über das Handy, rufen E-Mails mobil ab und spielen gegen andere Online-Spieler, die über die gesamte Welt verteilt sind. Diese zunehmende Technisierung unserer Welt führt dazu, dass Software immer mehr an Bedeutung gewinnt. In diesem Zuge kommt auch der Ausbildung im Software Engineering vermehrt Bedeutung zu.

Software Engineering beschäftigt sich mit der Entwicklung komplexer Softwaresysteme in einem großen, interdisziplinären Team über einen längeren Zeitraum hinweg und für eine mehr oder weniger bekannte Gruppe von späteren Anwendern. Häufig kann Software nicht nur von einem einzelnen Programmierer

entwickelt werden. Softwareentwicklung findet meist in einem oder mehreren Teams von Entwicklern statt. Gerade auch in dieser Komplexität liegen viele Tücken und Herausforderungen. Man denke nur an das vom amerikanischen Präsidenten Obama initiierte Gesundheitssystem „ObamaCare", dessen Einführung mit großen Schwierigkeiten behaftet war, was daran lag, dass zahlreiche Softwarefirmen an der Entwicklung der neuen Registrierungsplattform beteiligt und kaum zu koordinieren waren (Schmitt 2013). Auch wenn der Zusammenhang zwischen Projektgröße und -erfolg nicht zweifelsfrei belegbar ist (The Standish Group International 1999), scheitern viele Entwicklungsprojekte an zu engen Budget- und Zeitvorgaben oder an Änderungen von Zielsetzung und Anforderungen (Emam & Koru 2008). Softwareentwicklung ist ein sehr komplexer Prozess, der weit über das reine Programmieren hinausgeht. Softwareentwicklung beinhaltet zahlreiche Aktivitäten, die zum einen dazu führen, dass es innerhalb eines Entwicklungsteams eine Vielzahl an Rollen gibt, die ausgefüllt werden und zusammenarbeiten müssen. Zum anderen erfordert die Entwicklung von Software auch Kommunikation mit zahlreichen Stakeholdern und Kunden außerhalb des Teams (Vigenschow et al. 2011). Hinzu kommt, dass Software im Normalfall eben gerade nicht für Informatiker entwickelt wird, sondern für Anwender aus nahezu allen anderen Fachbereichen. Um in diesem Umfeld zu bestehen, benötigt ein Software-Ingenieur neben fundiertem Fachwissen auch zahlreiche überfachliche Kompetenzen. Erst diese sogenannten „weichen Kompetenzen" ermöglichen es dem Software-Ingenieur, sein fachliches Können in die Tat umzusetzen. Reines Fachwissen alleine genügt dafür nicht.

Um im späteren Berufsalltag bestehen zu können, ist es also erforderlich, in der Hochschulausbildung sowohl fachliches Können als auch überfachliche Kompetenzen zu adressieren.

Software Engineering weist eine Reihe von Besonderheiten auf, denen bereits in der Ausbildung Rechnung getragen werden muss: Neben umfassendem fachlichen Können sind besonders interpersonelle Kompetenzen von Bedeutung. Entgegen der weit verbreiteten Vorstellung, dass ein Informatiker alleine im dunklen Keller sitzt und programmiert, sind gerade im Software Engineering ausgeprägte kommunikative Fähigkeiten gefragt (Funken 2001). Auch die Zusammenarbeit in einem Team nimmt einen hohen Stellenwert ein und ein Software-Ingenieur muss sich in sehr große Teams von z.B. hundert Personen einfügen können.

1.2 Software Engineering im Unterricht der Hochschulen

Im Normalfall ist jedoch genau diese Komplexität in der Hochschulausbildung nur bedingt und in Ausschnitten abbildbar. Es sind meist nicht hundert Studierende vorhanden, die gemeinsam an einem Softwareprojekt arbeiten könnten, und auch keine Kunden, die selbst nicht so genau ausdrücken können, welche Anforderungen sie an eine zu entwickelnde Software haben. Ein Entwicklungsprojekt dauert normalerweise mehrere Monate bis Jahre. Auch dieses Zeitfenster mit den daraus resultierenden Herausforderungen, zum Beispiel sich verändernde Anforderungen und steigender Umfang des Softwareprodukts mit zunehmender Zeitdauer des Projektes, lässt sich im Hochschulkontext nicht abbilden. Denn zuerst muss über meh-

rere Semester hinweg ein Grundstock an fachlichem Wissen ausgebildet werden, bevor ein solches Entwicklungsprojekt überhaupt begonnen werden kann. Hinzu kommt, dass es sich bei Softwareentwicklung um einen extrem arbeitsteiligen Prozess mit einer Vielzahl an möglichen fachlichen Rollen und Aufgaben (Anforderungsanalytiker, Softwarearchitekt, Tester, etc.) handelt. Jeder beteiligte Ingenieur leistet einen wichtigen Beitrag, muss diesen aber wiederum an vielen Stellen mit Arbeitsergebnissen anderer vernetzen, diese aufgreifen und weiterentwickeln. Die Arbeit des einzelnen Entwicklers bringt nur dann Mehrwert, wenn sie mit der Arbeit der anderen verbunden wird. So macht es beispielsweise keinen Sinn, sich über die Struktur der zu entwickelnden Software oder mögliche Tests Gedanken zu machen, wenn nicht klar ist, was die Anforderungen an das zu entwickelnde System sind. Für Software-Ingenieure ist es manchmal schwierig, den Überblick über das Gesamtsystem zu behalten und den eigenen Beitrag als Teil dessen wahrzunehmen.

Da die Mehrzahl der Studierenden auch aufgrund der Bologna-Reformen und der Änderungen im Schulsystem (G8) noch relativ jung ist, fehlen meist persönliche Reife und berufliche Erfahrung. Diese wären aber nötig, um überhaupt ein Problembewusstsein für den Großteil der Herausforderungen im Software Engineering zu entwickeln. So können sich viele Studierende nicht vorstellen, dass die Anforderungen an eine zu entwickelnde Software nicht fix und fertig in einem Lastenheft vorliegen, sondern erst mühsam von den Kunden erfragt und in eine Sprache übersetzt werden müssen, mit der im Rahmen der Informatik weitergearbeitet werden kann. Viele Anwender selbst sehen hier auch kein Problem, denn sie erwarten, dass für das Problem, das sie den Informatikern meist knapp und nicht vollständig schildern, von diesen eine passgenaue Lösung geliefert würde. Vielen Studierenden ist nicht klar, dass beispielsweise Anforderungen sich über eine mehrmonatige Projektdauer mehrfach ändern oder immer neue hinzukommen können. Software Engineering erfordert ein hohes Maß an Austausch mit anderen wie z.B. mit späteren Anwendern, die zwar ein „Problem“ haben, es aber weder genau beschreiben können noch eine Lösung dafür kennen.

> Darüber hinaus haben die Kunden – in Unkenntnis der (Un-)Möglichkeiten von Softwarelösungen – meist sehr diffuse Vorstellungen von ihren Softwarewünschen und ändern diese häufig im Laufe der wachsenden Vertrautheit mit dem Thema (z.B. durch Prototypendemonstrationen). In der Praxis läßt sich deshalb eine der größten Schwierigkeiten der Entwickler mit der Frage formulieren: Was ist eigentlich das ‚Problem‘? (Funken 2001, S. 23)

Hinzu kommt, dass – selbst wenn Informatiker in der Lage sind, Abläufe im Unternehmen eines Kunden mittels geeigneter Modelle wie z.B. Prozessmodelle abzubilden – sie diese Abläufe zuerst vom Kunden erheben müssen. Häufig wurden die Fähigkeiten, um solche Informationen überhaupt zu bekommen, nicht im Studium adressiert. Dazu gehören zum Beispiel Arbeits-, Kreativitäts-, Kommunikations- und Fragetechniken.

Da jede Software nur einmal entwickelt wird, ist folglich auch jedes Entwicklungsprojekt anders. Für Softwareentwickler bedeutet dies, dass es keine Standard-

vorgehen oder Standardrezepte gibt. Jedes Entwicklungsprojekt ist einzigartig und erfordert neue, kreative Lösungsansätze.

Auf den Umgang mit genau diesen Herausforderungen sollte eine gute Hochschulausbildung vorbereiten. Dies begründet, Überlegungen in Richtung einer Fachdidaktik für Software Engineering anzustellen.

Zu Beginn der Informatikausbildung an Hochschulen lag der Schwerpunkt vorwiegend auf fachlichen Themen. Mittlerweile hat sich jedoch auch aufgrund der Softwarekrise in den 1960er und 70er Jahren häufig die Erkenntnis durchgesetzt, dass Fachwissen alleine nicht ausreichend für den Berufsalltag im Software Engineering qualifiziert. Ziel der Hochschulausbildung heute ist es, den Studierenden auch Problembewusstsein für die künftigen beruflichen Herausforderungen mitzugeben, ebenso wie das Wissen, was es bedeutet, Teil eines großen Entwicklungsteams zu sein, und dass es nicht genügt, gut programmieren zu können. Der Zusammenarbeit mit anderen kommt nun auch – gerade in der Hochschulausbildung von Software-Ingenieuren – eine besondere Bedeutung zu.

Diese Erkenntnis wirft jedoch neue Fragen auf:

- Was genau soll ein Software-Ingenieur bei Studienabschluss können? (Lehren wir das Richtige?)
- Welche (überfachlichen) Kompetenzen benötigt ein Software-Ingenieur?
- Was ist im Kontext von Software Engineering unter einer solchen überfachlichen Kompetenz zu verstehen? Was meint diese Kompetenz in genau diesem Zusammenhang? Was ist zum Beispiel gemeint, wenn von „kommunikativer Kompetenz" im Kontext von Software Engineering die Rede ist?
- Gibt es Zusammenhänge zwischen diesen Kompetenzen? Und wenn ja: Wie sehen diese aus?
- Sind die erforderlichen Kompetenzen gleich wichtig? Oder gibt es grundlegendere „Basiskompetenzen" und andere, die darauf aufbauen (Thurner et al. 2014)?

Dies alles zielt letztendlich auf die Frage ab, wie sich diese Kompetenzen in der Hochschulausbildung adressieren lassen (Lehren wir richtig?).

1.3 Vorhandene Konzepte

Erste Hinweise auf die erforderlichen Kompetenzen eines Software-Ingenieurs liefert der Guide to the Software Engineering Body of Knowledge, der mittlerweile in der dritten Überarbeitung erschienen ist (Bourque&Fairley 2014). Dieser ist als Standard anerkannt und beinhaltet verschiedene Wissensgebiete des Software Engineering.

Die derzeit aktuelle Version verfügt über einen Wissensbereich „Professional practice", der auch einzelne überfachliche Kompetenzen berücksichtigt. Dennoch reicht diese Richtlinie alleine nicht aus, um ein umfassendes Bild über die erforderlichen Kompetenzen im Software Engineering zu erhalten. Dafür gibt es mehrere Gründe. Zum einen ist nicht nachvollziehbar, woher die Daten und Wissensgebiete in SWEBOK kommen. Es ist kein nachvollziehbares Forschungsdesign beschrieben, das für die Gültigkeit der Daten sprechen würde. Jede sozialwissenschaftliche

Untermauerung fehlt. Zum anderen handelt es sich bei den Wissensgebieten in SWEBOK zunächst um eine reine Themensammlung. Zwar ist diese mit der Lernzieltaxonomie nach Bloom hinterlegt (Bloom 1972), jedoch liegt der Schwerpunkt auf fachlichem Wissen. Beim SWEBOK handelt es sich eher um eine fachliche Inhaltssammlung, die zwar eine Richtung für die Ausbildung geben kann, jedoch weder operationalisiert ist, noch sinnvolle Zusammenhänge zwischen den Themen anbietet. Größtes Manko sind jedoch die fehlenden überfachlichen Kompetenzen eines Software-Ingenieurs.

Die wenigen genannten überfachlichen Kompetenzen im SWEBOK werden zudem lediglich mit Schlagworten benannt, ohne nähere Definition, was damit genau gemeint sein könnte. Diesen Defiziten soll mit dem SWEBOS Rechnung getragen werden.

2 Software Engineering Body of Skills (SWEBOS)

2.1 Kompetenzmodell und Fragestellungen

Der „Software Engineering Body of Skills" (SWEBOS) ergänzt und vernetzt das SWEBOK mit überfachlichen Kompetenzen. SWEBOS stellt ein überfachliches Soll-Kompetenzprofil für Software Engineering nach Abschluss des Studiums dar.

Unsere Forschungsergebnisse legen nahe, dass eine Unterscheidung in fachliche Kompetenzen sowie kontextsensitive und allgemeine überfachliche Kompetenzen sinnvoll ist. Diese Unterscheidung ist notwendig, um den Fachbezug zum Software Engineering herzustellen und ein tiefes Verständnis (Flick et al. 2000) für die Bedeutung überfachlicher Kompetenzen im Kontext von Software Engineering zu erreichen. Die Unterscheidung in fachliche, kontextsensitive überfachliche und allgemeine überfachliche Kompetenzen stellt das zugrundeliegende pädagogische Kompetenzmodell dar. Ansätze in der empirischen Bildungsforschung plädieren dafür, allgemeine Schlüsselkompetenzen so weit zu konkretisieren, dass sie kontextsensitiv beschrieben werden und dann erst als Kompetenz bezeichnet werden sollten (Hartig 2008). Abweichend von dieser Vorstellung soll in diesem Kontext zwar von kontextsensitiven Kompetenzen gesprochen werden, jedoch sind auch allgemeine überfachliche Schlüsselqualifikationen in unserer Sichtweise Kompetenzen. Der Fokus in dieser Forschungsarbeit liegt auf den kontextsensitiven überfachlichen Kompetenzen eines Software-Ingenieurs. Fachliche Kompetenzen beziehen sich auf das Fachwissen eines Software-Ingenieurs, etwa das Beherrschen einer Programmiersprache oder die Kenntnis von Vorgehensmodellen. Hier lag jahrelang der Schwerpunkt der Hochschulausbildung, auch aus der Vorstellung heraus, dass sich die überfachlichen Kompetenzen von selbst einstellen. Fachwissen alleine reicht jedoch nicht aus, um als Software-Ingenieur tätig zu werden. Wie in jedem Beruf muss es um überfachliche Kompetenzen ergänzt werden, die während des Studiums auch berücksichtigt werden müssen.

Bei überfachlichen Kompetenzen lassen sich allgemeine überfachliche und kontextsensitive überfachliche Kompetenzen unterscheiden. Allgemeine überfachliche Kompetenzen können losgelöst vom fachlichen Kontext trainiert werden (z.B. Präsentationstechniken), während kontextsensitive überfachliche Kompetenzen sehr stark vom Fachlichen einer Disziplin, in diesem Fall des Software Engineering, charakterisiert werden. Kontextsensitive überfachliche Kompetenzen sollten im Zusammenhang mit den fachlichen Kompetenzen im Kontext von Software Engineering trainiert werden. Sie haben eine eigene Bedeutung im Kontext von Software Engineering, wie z.B. kommunikative Fähigkeiten oder Kompetenzen, in einem Team zu arbeiten. Was genau ist mit Worthülsen wie „Teamarbeit" oder „Kommunikative Kompetenz" im Kontext von Software Engineering gemeint? Wie prägen sie sich im speziellen Kontext von Software Engineering aus? Soll ein Software Entwickler drei verschiedene Sprachen beherrschen? Oder soll er in der Lage sein, ein Anforderungsdokument zu schreiben? Oder ist es vielmehr wichtig, dass er in der Lage ist, kommunikative Methoden und Techniken anzuwenden, die es ihm ermöglichen, Anforderungen zu erheben, zu präzisieren und zu dokumentieren? Benötigt er dafür nicht auch die Fähigkeit, interdisziplinär zu kommunizieren?

> Aushandlungsprozesse zwischen Partnern, die aus unterschiedlichen Handlungskontexten stammen, verlangen von den Beteiligten jedoch die Fähigkeit zum Perspektivenwechsel und die Anerkennung gleichwertiger Entscheidungskompetenzen. (Funken 2001, S. 15)

Die Fähigkeit, sich auf die Welt des jeweiligen Kunden oder Gesprächspartners einzulassen, ist eine Kernkompetenz im Software Engineering und stellt eine solche kontextsensitive überfachliche Kompetenz dar. Die Klärung, wie diese Kompetenz genau charakterisiert ist und wie ein erfolgreicher Software-Ingenieur agiert, der sie besitzt, ist Ziel dieses Forschungsansatzes.

Einzeln gesehen erscheinen kontextsensitive überfachliche Kompetenzen auf den ersten Blick wenig „besonders", bekommen jedoch bei näherer Betrachtung im Kontext der jeweiligen Fachlichkeit eine spezifische Charakteristik. Zudem bilden sie zusammen genommen in Verbindung mit fachlichen und allgemeinen überfachlichen Kompetenzen ein komplexes Geflecht, somit ein spezielles Kompetenzprofil für Software Engineering.

So erlernen Informatik-Studierende als fachliche Kompetenz zwar, wie man zum Beispiel einen Prozess modelliert, jedoch benötigen sie nicht nur die Notation, sondern zusätzlich Methoden und Techniken überfachlicher Natur, die es ihnen ermöglichen, die notwendigen zu modellierenden Informationen überhaupt erst zu bekommen. Dazu gehören zum Beispiel an den Kontext von Software Engineering angepasste, zielgerichtete Fragetechniken und die systematische Vorbereitung auf ein Kundengespräch genauso wie die Moderation dieses Kundengesprächs. Fragetechniken im Bereich des Software Engineering sind wesentlich durch den Kontext bestimmt, hier also die Gewinnung von Informationen zu Prozessen innerhalb einer Organisation und die dahinterliegende Modellvorstellung, was zur Beschreibung von Prozessen gehört. Fragen zielen also etwa auf die Ermittlung von Aktivitäten und die Ereignisse, die sie auslösen, um Anforderungen an ein Softwaresystem zu

erfassen. Die erfragten Informationen und somit auch die Fragetechniken sind darauf ausgerichtet, Informationen zu erhalten, die für die Entwicklung von Software wesentlich sind. Ein zentrales Ziel von Fragetechniken im Software Engineering ist zudem die Erhebung impliziter Anforderungen:

> Anstatt dem anderen aber die eigene handlungsleitende ‚Logik' zu enthüllen, vermittelt der Experte/Kunde zumeist nur Beispiele aus der Praxis oder seine subjektive Auswahl des Arbeitsprozesses und eventuell der Arbeitsumgebung. Selten nur ist er in der Lage, Prinzipien, nach denen er handelt, in Worte zu fassen. Die Entwickler müssen ihm also helfen, die impliziten Regeln seines Tuns zu erkennen bzw. das zu strukturieren, was er weiß. (Funken 2001, S. 138)

Daher erhalten allgemein scheinende kommunikative Kompetenzen wie etwa Fragetechniken im Kontext von Software Engineering eine spezielle Ausprägung und unterscheiden sich von Fragetechniken etwa in der Sozialen Arbeit. Eben genau diese Kompetenzen sind unabdingbar, damit technisches Fachwissen wie Modellierungsnotationen überhaupt erst anwendbar wird. Wenn ein Informatiker nicht in der Lage ist, die notwendigen Informationen vom Kunden zu erfragen, gibt es auch nichts, was als Prozessmodell dargestellt werden kann und woraus er Anforderungen ableiten könnte.

So sind etwa Fragetechniken auf den ersten Blick keine besondere kontextsensitive überfachliche Kompetenz. Es macht jedoch wenig Sinn, Informatikstudierende in Fragetechniken auszubilden, wenn der Anwendungsbezug, wie etwa der Einsatz in einem Kundengespräch zur Anforderungserhebung, fehlt. Fragetechniken erhalten eine besondere Ausprägung durch das Ziel, Anforderungen an ein Softwaresystem zu erheben. Zudem sind diese Kompetenzen mit weiteren fachlichen und überfachlichen Kompetenzen verknüpft, weil eine Fachkompetenz wie etwa eine Modellierungstechnik für Geschäftsprozesse die Art der zu erfragenden Information beeinflusst und der Frageprozess mittels Arbeitstechniken strukturiert werden muss. Hinzu kommt also neben dem Kontext die Kombination mit weiteren erforderlichen überfachlichen kontextsensitiven Kompetenzen wie etwa Arbeitstechniken.

Insgesamt stellt Software Engineering spezielle Anforderungen an die Entwickler, um z.B. Anforderungen zu erheben. Wie z.B. in einer anderen Fachdisziplin auch kommunikative Kompetenz gefordert ist, so müssen auch Software-Ingenieure die Brücke zwischen den verschiedenen Welten schlagen und die Inhalte in ihre eigene Informatiker-Fachwelt „übersetzen". Dies kann zudem individuell auf sehr unterschiedliche Art und Weise erfolgen und jeder Informatiker bringt seinen eigenen Stil und seine Persönlichkeit in den Arbeitsprozess ein:

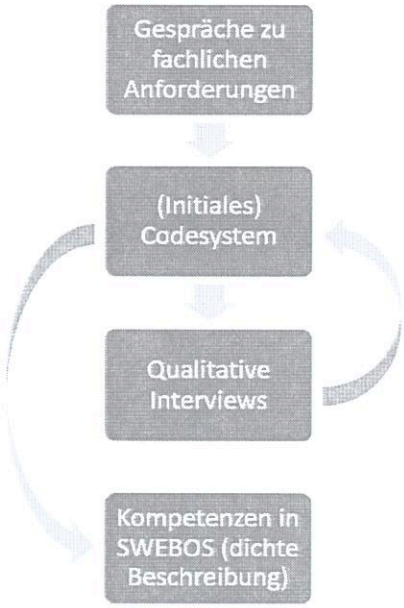
> Die relative Vielfalt der Software-Entwicklungsmodelle, die zahlreichen Programmiersprachen und -tools und der ‚universelle' Charakter des Computers ermöglichen im Prinzip eine weitgehende Ausdifferenzierung seiner Gebrauchsweisen. Software-Entwicklung wird gleichsam zum Vehikel der Stilbildung und Individuation; Ausbildung, Wissen, Talent, Persönlichkeit und organisatorische Bedingungen gerinnen zu einem je unverwechselbaren Entwicklungsstil. (Funken 2001, S. 61)

Das Ziel dieser Forschungsarbeit liegt darin herauszufinden, welche kontextsensitiven überfachlichen Kompetenzen im Software Engineering erforderlich sind, sowie ihre Ausprägungen und ihre Wechselwirkungen mit anderen Kompetenzen im Software Engineering zu beschreiben.

2.2 Methodische Vorgehensweise

Um diese Forschungsfrage zu beantworten, wurde ein qualitativer, datengetriebener Forschungsansatz zur Theoriebildung gewählt, der auf der Grounded Theory (Glaser &Strauss 2010) basiert.

Abb.1: Qualitatives Vorgehen



Ziel ist es, Kompetenzen eines Software-Ingenieurs bottom up zu verstehen und zu erklären, nicht nur zu beschreiben. Die Theorie wird generiert, nicht „nur" verifiziert. Ein Herzstück der Grounded Theory ist das Theoretische Sampling (Merkens 2000), was bedeutet, dass sich die Stichprobe abhängig von den Forschungsergebnissen weiterentwickelt und während des Forschungsprozesses entsteht. Bis zum Ende des Forschungsprozesses kann keine endgültige Aussage über die Zusammensetzung der Stichprobe getroffen werden (Glaser & Strauss 2010). Daraus resultiert auch, dass sich Datensammlung, -analyse und Theoriebildung über die gesamte Forschungsdauer hinweg ständig abwechseln und ineinander greifen. In einem ersten Schritt wurde daher die in Abb. 1 skizzierte qualitative Vorgehensweise gewählt.

Ausgangsmaterial waren zunächst Gespräche mit Software-Ingenieuren, die geführt wurden, um die fachlichen Inhalte des SWEBOK auf die Hochschulausbildung anzupassen und zu priorisieren. Diese Gespräche wurden nochmals ausgewertet, diesmal allerdings mit dem Fokus auf überfachlichen Kompetenzen. Dazu wur-

den in den transkribierten Interviews diejenigen Stellen codiert, in denen überfachliche Kompetenzen explizit oder implizit angesprochen wurden. Diese Codings wurden strukturiert und kategorisiert (siehe Abb. 1).

So wurde aus den vorhandenen ersten Daten ein erstes Codesystem entwickelt, das die Anforderungen an einen Software-Ingenieur strukturiert und clustert. Aufbauend auf diesem Codesystem wurde ein Interviewleitfaden entwickelt. Auf dieser Grundlage wurden Interviews mit Praktikern mit Personalverantwortung geführt. Diese Interviews erlauben über die Beschreibung des Berufsalltags eines Informatikers einen tieferen Einblick in die tatsächlich erforderlichen kontextsensitiven überfachlichen Kompetenzen eines Software-Ingenieurs. Sie eröffnen ein tieferes Verständnis, was wirklich damit gemeint ist, was diese Kompetenzen genau bedeuten und wie sie im Kompetenzprofil ineinandergreifen. Fragen an Berufspraktiker sind zum Beispiel:

- Beschreiben Sie bitte einen ganz normalen Arbeitstag, Ihre Aufgaben und Tätigkeiten.
- Was sind typische Herausforderungen / Stolpersteine in Ihrer täglichen Arbeit?
- Welche Faktoren sind Ihrer Meinung nach entscheidend für den Projekterfolg?
- Welche Fähigkeiten sind hier Ihrer Ansicht nach besonders wichtig? / Welche besonderen Fähigkeiten muss ein Software-Ingenieur haben?
- Warum halten Sie (genau) das für wichtig?
- Was erwarten Sie von einem guten Software-Ingenieur? Beschreiben Sie ihn bitte ein wenig.
- Wie bewerten Sie Kompetenzen in Personalauswahlgesprächen?
- Sie arbeiten im Bereich Software Engineering. Wie haben Sie das gelernt? Wie sind Sie ein guter Software-Ingenieur geworden?

Die Ergebnisse aus den Interviews führten zu einer Verfeinerung unseres Codesystems (siehe Abb. 2). Dabei stellte sich zur besseren Strukturierung eine Unterscheidung in intra- und inter-personelle Kompetenzen als sinnvoll heraus. Interpersonelle Kompetenzen beziehen sich auf Kompetenzen, die im Umgang und der Zusammenarbeit mit anderen Menschen besonders zum Tragen kommen, wie beispielsweise kommunikative Kompetenzen. Dagegen liegen intrapersonelle Kompetenzen in der Person und ihren Einstellungen und Werten selbst.

Abb. 2: Codesystem

Code-ID	Obercode	Code	Alle Codings	Alle Codings %	Dokumente
56	interpersonelle Kompetenzen	Kompromissfähigkeit	1	0,28	1
49	interpersonelle Kompetenzen	Innovationsfähigkeit/Energie	1	0,28	1
48	interpersonelle Kompetenzen	zuhören können	1	0,28	1
5	interpersonelle Kompetenzen	Zusammenarbeit	31	8,83	9
26	interpersonelle Kompetenzen	Kommunikation mit anderen (auch interdisziplinär)	18	5,13	5
11	interpersonelle Kompetenzen	Empathie	15	4,27	6
19	interpersonelle Kompetenzen	Einfügen in Organisationsstrukturen	13	3,70	5
18	interpersonelle Kompetenzen	Präsentation	11	3,13	7
24	interpersonelle Kompetenzen	Kritikfähigkeit	6	1,71	5
37	interpersonelle Kompetenzen	Kommunikation allgemein	4	1,14	3
14	interpersonelle Kompetenzen	Respektvoller Umgang	2	0,57	2
30	interpersonelle Kompetenzen	Konfliktlösung	2	0,57	2
54	interpersonelle Kompetenzen\Konfliktlösung	unangenehme Dinge vertreten können	1	0,28	1
43	intrapersonelle Kompetenzen	mentale/kognitive Fähigkeiten	0	0,00	0
42	intrapersonelle Kompetenzen	Umsetzungskompetenz	0	0,00	0
25	intrapersonelle Kompetenzen	Problembewußtsein	37	10,54	7
41	intrapersonelle Kompetenzen	Selbstkompetenz	0	0,00	0
40	intrapersonelle Kompetenzen	Umgang mit Stress/psychische Fähigkeiten	0	0,00	0
29	intrapersonelle Kompetenzen	sprachliche Fähigkeiten	8	2,28	5
20	intrapersonelle Kompetenzen	Verstehen komplexer Prozesse	31	8,83	7
35	intrapersonelle Kompetenzen	Arbeitsweise	0	0,00	0
16	intrapersonelle Kompetenzen\Arbeitsweise	Selbständigkeit	7	1,99	5
13	intrapersonelle Kompetenzen\Arbeitsweise	Zeitmanagement	1	0,28	1
12	intrapersonelle Kompetenzen\Arbeitsweise	Arbeitsorganisation	6	1,71	4
34	intrapersonelle Kompetenzen\Arbeitsweise	Gründlichkeit	1	0,28	1
31	intrapersonelle Kompetenzen\Arbeitsweise	methodisches Vorgehen/Systematik	4	1,14	2
17	intrapersonelle Kompetenzen\Arbeitsweise	Strukturierte Arbeitsweise	4	1,14	3
27	intrapersonelle Kompetenzen\mentale/kognitive Fähigkeiten	analytisches Denken	8	2,28	3
2	intrapersonelle Kompetenzen\mentale/kognitive Fähigkeiten	Abstraktionsvermögen	10	2,85	4
53	intrapersonelle Kompetenzen\Problembewußtsein	Problem erkennen können	2	0,57	1
33	intrapersonelle Kompetenzen\Problembewußtsein	Entscheidungsfähigkeit	2	0,57	2
55	intrapersonelle Kompetenzen\Selbstkompetenz	Begeisterungsfähigkeit	1	0,28	1
44	intrapersonelle Kompetenzen\Selbstkompetenz	realistische Selbsteinschätzung	1	0,28	1
32	intrapersonelle Kompetenzen\Selbstkompetenz	Zielstrebigkeit	5	1,42	3
7	intrapersonelle Kompetenzen\Selbstkompetenz	Offenheit für Neues / Flexibilität	14	3,99	4
10	intrapersonelle Kompetenzen\Selbstkompetenz	Selbst etwas erarbeiten können	16	4,56	6
23	intrapersonelle Kompetenzen\Selbstkompetenz	Selbstreflexion	4	1,14	3
8	intrapersonelle Kompetenzen\Selbstkompetenz	Lernbereitschaft (Gegenteil: Selbstüberschätzung)	22	6,27	6
6	intrapersonelle Kompetenzen\Umgang mit Stress/psychische Fähigkeiten	Frustrationstoleranz	8	2,28	4
28	intrapersonelle Kompetenzen\Umsetzungskompetenz	Problemlösungsfähigkeit/Kreativität	18	5,13	5
36	intrapersonelle Kompetenzen\Umsetzungskompetenz	Lerntransfer	1	0,28	1
15	intrapersonelle Kompetenzen\Verstehen komplexer Prozesse	Vorausschauendes Denken	2	0,57	2

Das Codesystem stellt einen Abstraktionsschritt dar und wurde bottom up aus den Daten heraus entwickelt. Es dient der Strukturierung der sogenannten dichten Beschreibung (Geertz 1987), die dazu dient, die Realität des Software Engineering und insbesondere die dort erforderlichen kontextsensitiven überfachlichen Kompetenzen zu verstehen und in einer sehr reichhaltigen und wenig abstrakten Darstellung, aber trotzdem allgemeingültig zu charakterisieren. Das Codesystem ist quasi ein Zwischenschritt zur „Ordnung" der Zusammenhänge, die dann wiederum in der dichten Beschreibung abgebildet werden.

2.3 Darstellungsform

Die Darstellungsform als dichte Beschreibung (Geertz 1987) soll zum einen eine strukturierte Übersicht über die erforderlichen kontextsensitiven überfachlichen Kompetenzen eines Software-Ingenieurs geben. Zum anderen soll sie auch ein Verständnis ermöglichen, was genau im Kontext von Software Engineering unter diesen Kompetenzen zu verstehen ist, und ein gesamtes, umfassendes Bild eines kompetenten Software-Ingenieurs ergeben. Daher beinhaltet die Darstellungsform in SWEBOS mehrere Ebenen:

- Benennung der Kompetenz auf relativ abstraktem Niveau
- Erläuterung
- Präzisierung, was an konkreten Handlungen damit verbunden sein sollte (Handlungsanker)

Ziel ist es nicht, einzelne Kompetenzen losgelöst vom Gesamtbild zu betrachten, sondern vielmehr steht das Verständnis für den Kontext Software Engineering im Fokus. Ziel ist ein umfassendes Abbild der Realität. Dabei sind die Handlungsanker als beispielhafter Richtwert zu verstehen, wie ein kompetenter Software-Ingenieur agieren sollte. Die individuelle Umsetzung, mit der die Kompetenz gezeigt wird, ist durch die Persönlichkeit der handelnden Person charakterisiert.

Zum einen soll das Kompetenzprofil SWEBOS sozialwissenschaftlichen Standards genügen und auch für die Pädagogik und verwandte Disziplinen einen Nutzen bringen. Zum anderen soll SWEBOS auch für technisch geprägte Fachdisziplinen als Kompass in der Hochschulausbildung dienen können und so eine praktische Verwendung erfahren. Die Darstellungsweise in Tabellenform wurde bewusst ausgewählt, um allen beteiligten Fachdisziplinen in diesem interdisziplinären Kontext gerecht zu werden. Während Sozial- und Humanwissenschaften sich vornehmlich mittels Prosatexten und sehr einfachen informellen Skizzen ausdrücken, sind technisch geprägte Disziplinen, zu denen auch die Informatik gehört, von Beginn der Ausbildung an darauf trainiert, sehr komplexe Sachverhalte zu abstrahieren und mittels bildlicher, semantisch definierter Darstellungsformen zu modellieren und zu strukturieren. Während pädagogisch geprägte Wissenschaftler häufig nach Erläuterungen und Erklärungen fragen, bitten Informatiker und Techniker meist um eine bildliche Modellierung des Sachverhalts. Um beiden Rechnung zu tragen und um SWEBOS für beide zugänglich zu machen, wurde die Tabellenform als Darstellungsweise ausgewählt.

3 Erste Ergebnisse

Die qualitative Auswertung der Interviews ergab folgende erforderlichen kontextsensitiven überfachlichen Kompetenzen, die ein Software-Ingenieur haben sollte:

Kompetenzen für die professionelle Zusammenarbeit mit anderen Menschen (Z)	
Diese Kompetenzen sind den anderen Kompetenzgruppen vorangestellt, da sie ein zentrales Grundelement für die Tätigkeit als Software-Ingenieur bilden. Nur wer in der Lage ist, mit anderen Menschen zusammenzuarbeiten, kann die an einen Software-Ingenieur gestellten Aufgaben lösen. Diese sind nicht ohne den organisatorischen Kontext zu stemmen. Unabhängig davon, welche Rolle ein Software-Ingenieur im jeweiligen Projekt einnimmt, muss er mit anderen Menschen zusammenarbeiten und seinen Beitrag zum großen Ganzen leisten.	
Z1	Software-Ingenieure können mit anderen in einem Team zusammenarbeiten.
Z2	Software-Ingenieure sind in der Lage und bereit, mit anderen, auch Fachfremden, zu kommunizieren.
Z3	Software-Ingenieure besitzen Empathie und sind in der Lage, sich in fremde Welten hineinzudenken.
Z4	Software-Ingenieure sind in der Lage und bereit, sich in bestehende Aufbau- und Ablauforganisationsstrukturen einzufügen.
Z5	Software-Ingenieure können anderen Menschen ihre Ideen und facheigenen Sachverhalte präsentieren.
Z6	Software-Ingenieure sind in der Lage, ihre eigene Fachlichkeit realistisch einzuschätzen, kennen ihre persönlichen Stärken und Schwächen und die Grenzen ihrer eigenen Fachlichkeit.
Z7	Software-Ingenieure erkennen in gleichem Maße die Fachlichkeit anderer an und pflegen einen respektvollen Umgang damit.

Kommunikative Kompetenzen (K)	
Aus einem ähnlichen Grund folgen den Kompetenzen für die professionelle Zusammenarbeit mit anderen Menschen die kommunikativen Kompetenzen eines Software-Ingenieurs. Sobald Menschen zusammenarbeiten, ist Kommunikation ein notwendiges und unvermeidbares Mittel des Informationsaustausches. Da jeder Mensch sein individuelles Modell der Welt besitzt und die an ihn herangetragenen Informationen darauf zugeschnitten filtert, sind Unklarheiten und Missverständnisse vorprogrammiert. Um diese Konflikte zu lösen bzw. nach Möglichkeit durch gezielte und angemessene Kommunikation gering zu halten, sind kommunikative Kompetenzen notwendig. Software-Ingenieure müssen in der Lage sein, andere Welten anzunehmen, zu akzeptieren, zu verstehen und darauf zu reagieren. Sie müssen bereit sein, sich auf fremde Weltbilder einzulassen und sie müssen in der Lage sein, damit umzugehen.	
K1	Software-Ingenieure sind in der Lage, mit Kritik umzugehen, d.h. sowohl angemessen ihre Meinung zu vertreten, in Diskussionen sachliche Beiträge zu liefern, als auch von anderen Feedback zu erhalten und dieses zu geben.
K2	Software-Ingenieure sind in der Lage, Konflikte konstruktiv zu lösen.

Kompetenzen, um die eigene Arbeit zu strukturieren (S)	
Diese Kompetenzen zielen darauf ab, dass Software-Ingenieure sich selbst und ihre eigene Arbeit in einem komplexen Umfeld strukturieren müssen. In einem extrem arbeitsteiligen Softwareentwicklungsprozess sind alle Beteiligten auf die Beiträge der anderen angewiesen, um ein funktionierendes Endprodukt herzustellen. Dazu müssen Software-Ingenieure sich selbst organisieren und strukturieren können.	
S1	Software-Ingenieure sind in der Lage, analytisch zu denken.
S2	Software-Ingenieure sind in der Lage, sich selbst Ziele zu setzen und darauf hinzuarbeiten.
S3	Software-Ingenieure können sich selbst motivieren, auch in komplexen Arbeitsabläufen und Teams ihren Beitrag über einen längeren Zeitraum zu leisten.
S4	Software-Ingenieure sind in der Lage, auch von sich aus, ohne Anstoß von außen, Aufgaben zu übernehmen und Probleme zu lösen.
S5	Software-Ingenieure sind in der Lage, ihre Zeit realistisch zu planen, Zeitpläne zu erstellen und danach die Aufgaben strukturiert abzuarbeiten.
S6	Software-Ingenieure sind gründlich in ihrer Arbeitsweise und erledigen ihre Aufgaben sorgfältig.

Personale Kompetenzen (P)	
Diese Kompetenzen zielen auf Selbstreflexion sowie den bewussten und zielgerichteten Umgang mit persönlichen Herausforderungen und Hindernissen.	
P1	Software-Ingenieure reflektieren sich, ihre Fähigkeiten und Fertigkeiten und ihr Verhalten regelmäßig und ziehen daraus Folgerungen für künftige Situationen.
P2	Software-Ingenieure sind in der Lage, auch unter Zeitdruck und in Stresssituationen ruhig und effizient zu arbeiten.
P3	Software-Ingenieure sind in der Lage, auch mit Rückschlägen umzugehen, diese auszuhalten und können angemessen damit umgehen. (Frustrationstoleranz)
P4	Software-Ingenieure sind sich bewusst, dass gelerntes Fachwissen erst in Kombination mit Erfahrung die Grundlage dafür ist, komplexe Softwaresysteme zu entwickeln.

Fähigkeit, komplexe Vorgänge und Systeme zu verstehen, Zusammenhänge zu verstehen (Problembewusstsein) (V)	
Diese Kompetenzen zielen auf das Abstrahieren komplexer Problemstellungen sowie auf das Konkretisieren von Lösungsmöglichkeiten. Gerade Software-Ingenieure kommen mit vielen anderen Disziplinen und Fachbereichen in Kontakt. Dies erfordert die Bereitschaft und Fähigkeit, über den Tellerrand hinauszuschauen und den Sinn und die Notwendigkeit vermeintlich ungeliebter Vorgehensweisen anzuerkennen und zu verstehen, warum manche Prozesse und Dokumente in einer bestimmten Art und Weise bzw. Form erforderlich sind.	
V1	Software-Ingenieure besitzen die Fähigkeit, komplexe Sachverhalte zu abstrahieren und zu modellieren.
V2	Software-Ingenieure erkennen, welches abstrakte Lösungsmuster auf die jeweilige Situation angewandt werden kann.

Fähigkeit, das eigene Wissen und Können, die eigenen Kompetenzen auf konkrete, neue Situationen flexibel und kreativ anzuwenden (Lösungskompetenz) (L)	
Das gelernte Fachwissen ist nicht in allen Bereichen vollständig und muss kreativ auf eine jeweilige Problemstellung angewandt und transferiert werden. Keine Problemstellung gleicht der anderen, so dass auch kein „Rezeptwissen" hilfreich ist. Vielmehr müssen über Abstraktion und das Bilden von Analogien gepaart mit Erfahrungswissen mögliche Lösungsmöglichkeiten entwickelt und analysiert werden. Dies erfordert zum einen Flexibilität, auf Herausforderungen schnell reagieren zu können und zum anderen die Fähigkeit, das eigene Wissen ständig weiterzuentwickeln und zu aktualisieren, also lebenslang zu lernen.	
L1	Software-Ingenieure sind in der Lage, mögliche Lösungswege für fachliche Problemstellungen kreativ zu entwickeln.
L2	Software-Ingenieure sind in der Lage, verschiedene Lösungswege zu bewerten, abzuwägen, sich für einen möglichen Lösungsweg zu entscheiden und diesen auch umzusetzen.
L3	Software-Ingenieure sind in der Lage und bereit, sich selbständig in für sie neue Themen und Bereiche einzuarbeiten und bereit, lebenslang selbständig neues zu lernen.
L4	Software-Ingenieure sind offen für neues und andere. Sie sind in der Lage und bereit, sich flexibel auf unvorhergesehene und unplanbare Situationen einzulassen und darauf zu reagieren.

Weitere Kompetenzen (W)	
W1	Software-Ingenieure übernehmen Verantwortung für andere und ein gemeinsames Projekt.
W2	Software-Ingenieure sind in der Lage, nach benötigten Informationen zu suchen, diese zu finden und für die aktuelle Fragestellung weiterzuverwenden.
W3	Software-Ingenieure sind in der Lage, die Folgen des eigenen Handelns abzuschätzen und handeln nach gesellschaftlichen und ethischen Grundsätzen.
W4	Software-Ingenieure sind in der Lage, sich schriftlich auszudrücken.

4 Einbettung in Forschungskontext und nächste Schritte

Gemeinsam mit den vorwiegend fachlichen Kompetenzen des SWEBOK bietet SWEBOS eine Orientierung, welche Kompetenzen in der Software Engineering Ausbildung an Hochschulen berücksichtigt werden sollten.

Im Rahmen des Forschungsprojektes „Experimentelle Verbesserung des Lernens von Software Engineering" (EVELIN) werden erste Schritte in Richtung einer Fachdidaktik für Software Engineering gegangen (Figas et al. 2014). Dazu werden didaktische Ansätze vorhandener Lehrveranstaltungen analysiert und systematisch im Hinblick auf die zu trainierenden Kompetenzen des Software Engineerings weiterentwickelt (Sedelmaier & Landes 2013). In einem ersten Schritt werden die Lehrziele verschiedener Lehrveranstaltungen mit den Soll-Kompetenzprofilen SWEBOS und SWEBOK abgeglichen und gegebenenfalls justiert. Dann erfolgt darauf aufbauend eine Überprüfung didaktischer Methoden und Ansätze hinsichtlich der Passgenauigkeit, ob die didaktische Umsetzung die angestrebten Ziele for-

ciert. Damit sollen in der Hochschulausbildung des Software Engineering zusätzlich zum Fachwissen auch kontextsensitive überfachliche Kompetenzen adressiert und so die Lehre kompetenzorientiert weiterentwickelt werden (Sedelmaier & Landes 2014a, 2014b, 2014c, Abke et al. 2013; Landes et al. 2012).

Im weiteren Verlauf des Forschungsprojektes soll das Kompetenzprofil SWEBOS noch weiter erforscht werden. So ist angedacht, qualitative Interviews mit weiteren Personengruppen wie etwa Lehrenden, Absolventen oder Software-Ingenieuren (ohne Personalverantwortung) zu führen und auszuwerten.

Auch ist denkbar, die Gültigkeit des Kompetenzprofils SWEBOS in Nachbardisziplinen zu überprüfen und zu verfeinern, die ebenfalls Softwareentwicklung betreiben. SWEBOS berücksichtigt derzeit die kontextsensitiven überfachlichen Kompetenzen von Software-Ingenieuren. Jedoch wird sowohl in der „klassischen" Informatik als auch in verwandten Disziplinen wie etwa der Wirtschaftsinformatik, der Mechatronik oder der Elektrotechnik Software entwickelt. Softwaresysteme in technischen Disziplinen, so genannte eingebettete Systeme, sind meist sehr hardwarenah und beinhalten keine direkte Interaktion mit Anwendern, wie etwa ein Airbag im Auto. Hier unterscheiden sich die Entwicklungsprozesse an manchen Punkten von der klassischen Informatik, weisen aber auch in vielen Bereichen große Ähnlichkeit auf. Solche Entwicklungsprojekte sind etwa genauso komplex und erfordern ebenso Kommunikation innerhalb eines Teams und mit anderen Fachdisziplinen. Eine Überprüfung, ob und wie sich die Schwerpunkte des entstandenen Kompetenzprofils SWEBOS in den verschiedenen Disziplinen, die Software Engineering betreiben, unterscheiden. Der Abgleich mit schon vorhandenen Forschungsergebnissen (Gold et al. 2014) bietet möglicherweise auch hier Ansätze für die Weiterentwicklung der Hochschulausbildung.

Ein weiterer Fokus der künftigen Forschung könnte in der genaueren Erforschung von Zusammenhängen von Kompetenzen liegen. Es liegen noch keine genaueren Erkenntnisse vor, ob Teilkompetenzen auf anderen Basiskompetenzen aufbauen, welche Rolle der Zusammenhang mit fachlichen Kompetenzen unter diesem Gesichtspunkt spielt und welches eher grundlegende bzw. aufbauende Kompetenzen sind.

Um Aussagen zur Wirksamkeit (fach)didaktischer Ansätze treffen zu können, wäre es hilfreich, Kompetenzen beziehungsweise ihre Entwicklung möglichst valide und reliabel einschätzen zu können. Zur Kompetenzmessung gibt es zahlreiche vielversprechende Ansätze, besonders in der beruflichen Bildung (Rauner 2011). Derzeit wird im Forschungsprojekt EVELIN[1] ausgehend von einem solchen Ansatz ein Bewertungsverfahren entwickelt, das helfen soll, Kompetenzen im Software Engineering aus verschiedenen Perspektiven zu bewerten, und so Rückschlüsse auf didaktische Designs erlauben soll (Sedelmaier & Landes 2014a).

1 Das Forschungsprojekt EVELIN wird aus Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01PL12022A gefördert

Literatur

Abke, J.; Schwirtlich, V.; Sedelmaier, Y. (2013): Kompetenzförderung im Software Engineering durch ein mehrstufiges Lehrkonzept im Studiengang Mechatronik. In A. Schmolitzky, D. Rick und P. Forbrig (Hrsg.): *HDI 2012 – Informatik für eine nachhaltige Zukunft. 5. Fachtagung zur Hochschuldidaktik der Informatik*, 06. – 07. November 2012, Universität Hamburg. Potsdam: Universitäts-Verlag Potsdam (Commentariiinformaticaedidacticae (CID)), S. 79–84.

Bloom, B. S. (1972): *Taxonomie von Lernzielen im kognitiven Bereich*. Weinheim: Beltz.

Bourque, P.; Fairley, R. E. (Hrsg.) (2014): *Guide to the Software Engineering Body of Knowledge Version 3.0 – SWEBOK*. IEEE Computer Society. Online verfügbar unter http://www.computer.org/ieeecs-swebokdelivery-portlet/swebok/SWEBOKv3.pdf?token=RkjKSKMu0hA2hyGQxx5lHHB1LTkirNPu, zuletzt geprüft am 13.03.2014.

Emam, K. E.; Koru, A. G. (2008): A Replicated Survey of IT Software Project Failures. *IEEE Softw. 25* (5), 84–90.

Figas, P.; Knörl, S.; Mörtlbauer, S.; Sedelmaier, Y.; Schroll-Decker, I. (2014): Software Engineering Education As a Didactical Discipline. In *1st European Conference on Software Engineering Education (ECSEE)*, im Erscheinen.

Flick, U.; von Kardorff, E.; Steinke, I. (2000): Was ist qualitative Forschung? Einleitung und Überblick. In E. von Kardorff, I. Steinke und U. Flick (Hrsg.): *Qualitative Forschung. Ein Handbuch*. Reinbek bei Hamburg: Rowohlt, S. 13–29.

Funken, C. (2001): *Die Modellierung der Welt. Wissenssoziologische Studien zur Software-Entwicklung*. Opladen: Leske + Budrich.

Geertz, C. (1987): *Dichte Beschreibung. Beiträge zum Verstehen kultureller Systeme*. 12. Aufl. Frankfurt a.M., Germany: Suhrkamp (Suhrkamp-Taschenbuch Wissenschaft).

Glaser, B. G.; Strauss, A. L. (2010): *Grounded theory. Strategien qualitativer Forschung*. 3. Aufl. Bern: Huber.

Gold, C.; Sedelmaier, Y.; Abke, J. (2014): A Retrospective Course Survey of Graduates to Analyse Competencies in Software Engineering. In *Global Engineering Education Conference (EDUCON)*: IEEE, S. 100–106.

Hartig, J. (2008): Kompetenzerfassung von Bildungsprozessen. In N. Jude, J. Hartig und E. Klieme (Hrsg.): *Kompetenzerfassung in pädagogischen Handlungsfeldern – Theorien, Konzepte und Methoden*. Bonn, Berlin: Bundesministerium für Bildung und Forschung (BMBF) (Bildungsforschung, 26), S. 15–25.

Landes, D.; Sedelmaier, Y.; Pfeiffer, V.; Mottok, J.; Hagel, G. (2012): Learning and teaching software process models. In *Global Engineering Education Conference (EDUCON)*, S. 1153–1160. Online verfügbar unter http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6201191.

Merkens, H. (2000): Auswahlverfahren, Sampling, Fallkonstruktion. In E. von Kardorff, I. Steinke und U. Flick (Hrsg.): *Qualitative Forschung. Ein Handbuch*. Reinbek bei Hamburg: Rowohlt, S. 286–299.

Rauner, F. (2011): *Messen beruflicher Kompetenzen*. Münster: Lit.

Schmitt, U. (2013): Obamacare entpuppt sich als gigantischer Flop. In *Die Welt*, 13.11.2013. Online verfügbar unter http://www.welt.de/politik/ausland/article 121855210/Obamacare-entpuppt-sich-als-gigantischer-Flop.html, zuletzt geprüft am 12.03.2014.

Sedelmaier, Y.; Landes, D. (2013): A Research Agenda for Identifying and Developing Required Competencies in Software Engineering. In *International Journal of Engineering Pedagogy (iJEP), 3* (2), 30–35.

Sedelmaier, Y.; Landes, D. (2014a): A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies. In *44th Frontiers in Education (FIE)*: IEEE, im Erscheinen.

Sedelmaier, Y.; Landes, D. (2014b): Practicing Soft Skills in Software Engineering. In L. Yu (Hrsg.): *Overcoming Challenges in Software Engineering Education: IGI Global*, S. 161–179.

Sedelmaier, Y.; Landes, D. (2014c): Software Engineering Body of Skills. In *Global Engineering Education Conference (EDUCON)*: IEEE, S. 395–401.

Sedelmaier, Y.; Landes, D. (2014d): Using Business Process Models to Foster Competencies in Requirements Engineering. In *27th International Conference on Software Engineering Education and Training (CSEE&T)*. Klagenfurt, April 23-25, S. 13–22.

Sommerville, I. (2011): Software engineering. 9th ed. Boston: Pearson.

The Standish Group International (1999): CHAOS: *A Recipe for Success. The Standish Group International*. Online verfügbar unter https://www4.informatik.tu-muenchen.de/lehre/vorlesungen/vse/WS2004/1999_Standish_Chaos.pdf, zuletzt geprüft am 05.05.2014.

Thurner, V.; Böttcher, A.; Kämper, A. (2014): Identifying Base Competencies as Prerequisites for Software Engineering Education. In *Global Engineering Education Conference (EDUCON)*: IEEE, S. 1069–1076.

Vigenschow, U.; Schneider, B.; Meyrose, I. (2011): *Soft Skills für Softwareentwickler. Fragetechniken, Konfliktmanagement, Kommunikationstypen und -modelle*. 2. Aufl. Heidelberg: dpunkt.

Anhang 2: Übersicht Copyright-Status

Publikation	Copyrightstatus
Anhang 1.1 Sedelmaier, Y. & Landes, D. (2014). A Multi-Level Didactical Approach to Build up Competencies in Requirements Engineering. In B. Penzenstadler, S. Gregory & D. Landes (Hrsg.), *8th International Workshop on Requirements Engineering Education & Training (REET 2014), 22nd International Conference on Requirements Engineering (RE 2014)* (Bd. 1217, S. 26–34). CEUR Workshop Proceedings vol. 1217. Zugriff am 04.09.2014. Verfügbar unter http://ceur-ws.org/Vol-1217/paper4.pdf.	Copyright für Verwendung in der Dissertation erteilt
Anhang 1.2 Sedelmaier, Y. & Landes, D. (2014). A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies. SECAT - A Software Engineering Competency Assessment Tool. In IEEE (Hrsg.), *44th Frontiers in Education (FIE)* (S. 2065–2072)	Copyright für Verwendung in der Dissertation erteilt
Anhang 1.3 Sedelmaier, Y. & Landes, D. (2014). Using Business Process Models to Foster Competencies in Requirements Engineering. In *27th International Conference on Software Engineering Education and Training (CSEE&T)* (S. 13–22)	Copyright für Verwendung in der Dissertation erteilt
Anhang 1.4 Sedelmaier, Y. & Landes, D. (2015). Active and Inductive Learning in Software Engineering Education. In *37th International Conference on Software Engineering (ICSE)* (S. 418–427)	Copyright für Verwendung in der Dissertation erteilt
Anhang 1.5 Sedelmaier, Y. & Landes, D. (2015). Towards a Better Understanding of Learning Mechanisms in Information Systems Education. A Competence-Oriented Approach to Subject-Matter Didactics. In IEEE (Hrsg.), *Global Engineering Education Conference (EDUCON)* (S. 418–427). IEEE.	Copyright für Verwendung in der Dissertation erteilt
Anhang 1.6 Sedelmaier, Y. & Landes, D. (2015). Überfachliche Kompetenz im Software Engineering - Modellierung, Förderung und Messung in der Hochschulausbildung. In U. Riegel, S. Schubert, G. Siebert-Ott & K. Macha (Hrsg.), *Kompetenzmodellierung und -messung in den Fachdidaktiken* (S. 111–130). Münster: Waxmann.	Für die Dissertation genehmigter Sonderdruck

Anhang 3: Publikationsliste

Publikationen Yvonne Sedelmaier

Stand: 04.12.2015

Anmerkungen:

a) Sofern vorhanden wird am Ende jedes Verweises auf den wissenschaftlichen Impact (I) verwiesen. Dabei liegt http://core.edu.au/index.php/conference-rankings zugrunde.[1]
b) Erklärung zu den Eigenanteilen: Bei einer Hauptautorenschaft wird „HA" vermerkt, bei Alleinautorenschaft „AA"; des Weiteren wird die Beteiligung an der Konzeption (K), der theoretischen Begründung des Ansatzes (T) sowie das Strukturieren und Schreiben des Ansatzes (S) anteilig vermerkt.
c) Alle aufgeführten Publikationen wurden peer-reviewed.

2015

Sedelmaier, Y. & Landes, D. (2015). Towards a Better Understanding of Learning Mechanisms in Information Systems Education. A Competence-Oriented Approach to Subject-Matter Didactics. In IEEE (Hrsg.), *Global Engineering Education Conference (EDUCON)* (S. 418–427). IEEE.
I: --; HA, K: 60%, T: 100%, S: 50%; (best-paper-award)

Sedelmaier, Y. & Landes, D. (2015). A Competence-Oriented Approach to Subject-Matter Didactics for Software Engineering. *International Journal of Engineering Pedagogy (iJEP), 5*(3), 34–44.
I: --; HA, K: 60%, T: 100%, S: 50%;

Sedelmaier, Y. & Landes, D. (2015). Ein mehrstufiges Lehrkonzept für RE. In D. W. Cunningham (Hrsg.), *Informatik 2015. Tagung vom 28. September – 2. Oktober 2015 in Cottbus* (GI-Edition Lecture Notes in Informatics Proceedings, Bd. 246, S. 619–620). Bonn: Köllen.
I: --; HA, K: 50%, T: 100%, S: 50%

Sedelmaier, Y. & Landes, D. (2015). Active and Inductive Learning in Software Engineering Education. In *37th International Conference on Software Engineering (ICSE)* (S. 418–427)
I: A*; HA, K: 50%, T: 100%, S: 50%

[1] Conferences are assigned to one of the following categories:

- A* - flagship conference, a leading venue in a discipline area
- A - excellent conference, and highly respected in a discipline area
- B - good conference, and well regarded in a discipline area
- C - other ranked conference venues that meet minimum standards
- Australasian - A conference for which the audience is primarily Australians and New Zealanders
- Unranked - A conference for which no ranking decision has been made

Sedelmaier, Y. & Landes, D. (2015). Überfachliche Kompetenz im Software Engineering - Modellierung, Förderung und Messung in der Hochschulausbildung. In U. Riegel, S. Schubert, G. Siebert-Ott & K. Macha (Hrsg.), *Kompetenzmodellierung und -messung in den Fachdidaktiken* (S. 111–130). Münster: Waxmann.
I: --; AA, K: 100%, T: 100%, S: 100%;

Sedelmaier, Y. & Landes, D. (2015). SWEBOS - The Software Engineering Body of Skills. *International Journal of Engineering Pedagogy (iJEP), 5*(1), 12–19.
I: --; AA, K: 100%, T: 100%, S: 100%;

2014

Sedelmaier, Y. & Landes, D. (2014). Using Business Process Models to Foster Competencies in Requirements Engineering. In *27th International Conference on Software Engineering Education and Training (CSEE&T)* (S. 13–22)
I: C; HA, K: 50%, T: 100%, S: 50%;

Sedelmaier, Y. & Landes, D. (2014). Practicing Soft Skills in Software Engineering. In L. Yu (Hrsg.), *Overcoming Challenges in Software Engineering Education* (S. 161–179). IGI Global.
I: --; HA, K: 60%, T: 100%, S: 60%;

Gold, C., Sedelmaier, Y. & Abke, J. (2014). A Retrospective Course Survey of Graduates to Analyse Competencies in Software Engineering. In IEEE (Hrsg.), *Global Engineering Education Conference (EDUCON)* (S. 100–106). IEEE.
I: --; K: 15%, T: 20%, S: 15%;

Sedelmaier, Y. & Landes, D. (2014). A Multi-Level Didactical Approach to Build up Competencies in Requirements Engineering. In B. Penzenstadler, S. Gregory & D. Landes (Hrsg.), *8th International Workshop on Requirements Engineering Education & Training (REET 2014), 22nd International Conference on Requirements Engineering (RE 2014)* (Bd. 1217, S. 26–34). CEUR Workshop Proceedings vol. 1217. Zugriff am 04.09.2014. Verfügbar unter http://ceur-ws.org/Vol-1217/paper4.pdf.
I: A; HA, K: 50%, T: 100%, S: 50%;

Sedelmaier, Y. & Landes, D. (2014). A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies. SECAT - A Software Engineering Competency Assessment Tool. In IEEE (Hrsg.), *44th Frontiers in Education (FIE)* (S. 2065–2072)
I: B; HA, K: 90%, T: 100%, S: 80%;

Figas, P., Knörl, S., Mörtlbauer, S., Sedelmaier, Y. & Schroll-Decker, I. (2014). Developing Software Engineering Education as a Didactical Discipline in its own right. In G. Hagel & J. Mottok (Hrsg.), *1st European Conference Software Engineering Education (ECSEE)* (S. 1–15). Aachen: Shaker.
I: --; K: 25%, T: 25%, S: 25%; (best-paper-award)

Sedelmaier, Y. & Landes, D. (2014). Software Engineering Body of Skills. In IEEE (Hrsg.), *Global Engineering Education Conference (EDUCON)* (S. 395–401). IEEE.
I: --; AA, K: 100%, T: 100%, S: 100%;

2013

Sedelmaier, Y. & Landes, D. (2013). A Research Agenda for Identifying and Developing Required Competencies in Software Engineering. *International Journal of Engineering Pedagogy (iJEP), 3.*(2), 30–35.
I: --; HA, K: 80%, T: 100%, S: 80%

Abke, J., Schwirtlich, V. & Sedelmaier, Y. (2013). Kompetenzförderung im Software Engineering durch ein mehrstufiges Lehrkonzept im Studiengang Mechatronik. In A. Schmolitzky, D. Rick & P. Forbrig (Hrsg.), *HDI 2012 – Informatik für eine nachhaltige Zukunft. 5. Fachtagung zur Hochschuldidaktik der Informatik, 06. – 07. November 2012, Universität Hamburg* (Commentarii informaticae didacticae (CID), S. 79–84). Potsdam: Universitäts-Verlag Potsdam.
I: --; HA, K: 20%, T: 100%, S: 50%;

Sedelmaier, Y., Claren, S. & Landes, D. (2013). Welche Kompetenzen benötigt ein Software Ingenieur? In A. Spillner & H. Lichter (Hrsg.), *Software Engineering im Unterricht der Hochschulen 2013* (S. 117–128). Zugriff am 29.06.2013. Verfügbar unter http://ceur-ws.org/Vol-956/S4_Paper4.pdf.
I: --; HA, K: 50%, T: 100%, S: 50%;

Abke, J., Gold, C., Roznawski, N., Schwirtlich, V. & Sedelmaier, Y. (2013). A new approach to collaborative learning in software engineering focussed on embedded systems. In *International Conference on Interactive Collaborative Learning (ICL)* (S. 610–616)
I: --; K: 20%, T: 80%, S: 30%;

2012

Sedelmaier, Y. & Landes, D. (2012). A Research Agenda for Identifying and Developing Required Competencies in Software Engineering. In *15th International Conference on Interactive Collaborative Learning (ICL); 41st International Conference on Engineering Pedagogy;*
I: --; HA, K: 80%, T: 100%, S: 80%

Landes, D., Sedelmaier, Y., Pfeiffer, V., Mottok, J. & Hagel, G. (2012). Learning and teaching software process models. In *Global Engineering Education Conference (EDUCON)* (S. 1153–1160).
I: --; K: <20%, T: >20%, S: >20% (best-paper-award)

Claren, S. & Sedelmaier, Y. (2012). Ein Kompetenzrahmenmodell für Software Engineering. Ein Schema zur Beschreibung von Kompetenzen. In *Embedded Software Engineering (ESE)* (S. 647–652)
I: --; HA, K: 50%, T: 100%, S: 30%

Abke, J., Brune, P., Haupt, W., Hagel, G., Landes, D., Mottok, J. et al. (2012). EVELIN – ein Forschungsprojekt zur systematischen Verbesserung des Lernens von Software Engineering. In *Embedded Software Engineering (ESE)* (S. 653–658)
I: --; HA, K: >25%, T: >35%, S: <20%

Vorträge:

Sedelmaier, Yvonne; Landes, Dieter: Active and Inductive Learning in Software Engineering Education. 37th International Conference on Software Engineering (ICSE). Florenz, Italien. IEEE, 21.05.2015.

Sedelmaier, Yvonne; Landes, Dieter (2014): A Multi-Level Didactical Approach to Build up Context-Sensitive Competencies in Requirements Engineering. 8th International Workshop on Requirements Engineering Education and Training (REET 2014). 22nd International Requirements Engineering Conference (RE). IEEE. Karlskrona, Schweden, 25.08.2014.

Sedelmaier, Yvonne: Using Business Process Models to Foster Competencies in Requirements Engineering. Alpen-Adria-Universität. IEEE Computer Society. Klagenfurt, Österreich, 24.04.2014.

Sedelmaier, Yvonne: Software Engineering Body of Skills (SWEBOS). IEEE Computer Society. EDUCON 2014. Istanbul, Türkei, 05.04.2014.

Sedelmaier, Yvonne: Entwicklung eines Leitkonzepts für die Hochschuldidaktische Lehre von Software Engineering. 43. dghd Jahrestagung 2014: Leitkonzepte der Hochschuldidaktik: Theorie - Praxis - Empirie. Deutsche Gesellschaft fürh Hochschuldidaktik (dghd). TU Braunschweig, 18.03.2014.

Sedelmaier, Yvonne (2013): Überfachliche Kompetenzen im Software Engineering. Modellierung, Förderung und Messung in der Hochschulausbildung. Universität Siegen. Siegener Zentrum für Lehrerbildung und Bildungsforschung, Geschäftsstelle Bildungsforschung. Siegen, 12.09.2013.

Sedelmaier, Yvonne (2012): Kompetenzförderung im Software Engineering - Ein mehrstufiges Lehrkonzept im Studiengang Mechatronik. Gesellschaft für Informatik e.V. (GI). HDI 2012 - Informatik für eine nachhaltige Zukunft; 5. Fachtagung Hochschuldidaktik der Informatik. Universität Hamburg, 06.11.2012.

Sedelmaier, Yvonne (2012): A Research Agenda for Identifying and Developing Required Competencies in Software Engineering. IEEE Computer Society. ICL/IGIP. Villach, Österreich, 27.09.2012.

Anhang 4: Erklärungen zu den Eigenanteilen

Erklärung zu den Eigenanteilen des Promovenden/der Promovendin sowie der Koautoren/-innen an den Publikationen bei einer kumulativen Dissertation

Erläuterung:
Der Anteil des eigenen Beitrags im Verhältnis zum Beitrag der anderen Ko-Autoren oder Ko-Autor-innen an der Erstellung des Artikels ist insgesamt unter angemessener Berücksichtigung aller Arten von Beiträgen nach folgender Klassifizierung anzugeben:

- Allein-Autorenschaft, wenn der eigene Anteil 100% beträgt.
- Überwiegender Anteil, wenn der eigene Anteil den Anteil jedes einzelnen anderen Ko-Autoren und jeder einzelnen anderen Ko-Autorin überwiegt und mindestens 35% beträgt.
- Gleicher Anteil, wenn (1) der eigene Anteil gleich hoch ist wie der von anderen Ko-Autoren oder Ko-Autorinnen, (2) kein weiterer Ko-Autor und keine weitere Ko-Autorin einen Anteil hat, der größer ist als der eigene, und (3) der eigene Anteil mindestens 25% beträgt.
- Wichtiger Anteil, wenn der eigene Anteil mindestens 20% beträgt, aber nicht Allein-Autorenschaft, überwiegenden Anteil oder gleichen Anteil darstellt.
- Geringer Anteil, wenn der eigene Anteil weniger als 20% beträgt.

2015

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2015): Towards a Better Understanding of Learning Mechanisms in Information Systems Education. A Competence-Oriented Approach to Subject-Matter Didactics. In: IEEE (Hg.): Global Engineering Education Conference (EDUCON): Tallinn, Estland, S. 418–427. (peer reviewed) (best-paper-award)

Konzeption des beschriebenen Ansatzes	Überwiegender Anteil (ca. 60 %)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Gleicher Anteil (ca. 50 %)

Sedelmaier, Yvonne

Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2015): A Competence-Oriented Approach to Subject-Matter Didactics for Software Engineering. In: International Journal of Engineering Pedagogy (iJEP), H. 3, 34–44. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Überwiegender Anteil (ca. 60%)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Gleicher Anteil (50%)

Sedelmaier, Yvonne Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2015): Ein mehrstufiges Lehrkonzept für RE. In: D. W. Cunningham (Hrsg.), *Informatik 2015. Tagung vom 28. September – 2. Oktober 2015 in Cottbus* (GI-Edition Lecture Notes in Informatics Proceedings, Bd. 246, S. 619–620). Bonn: Köllen. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Gleicher Anteil (50%)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Gleicher Anteil (50%)

Sedelmaier, Yvonne Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2015): Active and Inductive Learning in Software Engineering Education. In: IEEE (Hg.): 37th International Conference on Software Engineering (ICSE): Florenz, Italien, S 418-427. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Gleicher Anteil (ca. 50 %)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Gleicher Anteil (ca. 50 %)

Sedelmaier, Yvonne Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2015): Überfachliche Kompetenzen im Software Engineering - Modellierung, Förderung und Messung in der Hochschulausbildung. In: Ulrich Riegel, Sigrid Schubert, Gesa Siebert-Ott und Klaas Macha (Hg.): Kompetenzmodellierung und -messung in den Fachdidaktiken. Münster: Waxmann, S. 111–130. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Allein-Autorenschaft (100%)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Allein-Autorenschaft (100%)

Sedelmaier, Yvonne

Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2015): SWEBOS - The Software Engineering Body of Skills. In: International Journal of Engineering Pedagogy (iJEP), H. 1, 12–19. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Allein-Autorenschaft (100%)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Allein-Autorenschaft (100%)

Sedelmaier, Yvonne

Landes, Dieter

Unterschriften der AutorInnen

2014

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2014): Using Business Process Models to Foster Competencies in Requirements Engineering. In: IEEE (Hg.): 27th International Conference on Software Engineering Education and Training (CSEE&T): Klagenfurt, Österreich, S. 13–22. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Gleicher Anteil (ca. 50 %)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Gleicher Anteil (ca. 50 %)

Sedelmaier, Yvonne

Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2014): Practicing Soft Skills in Software Engineering. In: Liguo Yu (Hg.): Overcoming Challenges in Software Engineering Education: IGI Global, S. 161–179. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Überwiegender Anteil (ca. 60 %)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Überwiegender Anteil (ca. 60 %)

Sedelmaier, Yvonne

Landes, Dieter

Unterschriften

Publikation:

Gold, Carolin; Sedelmaier, Yvonne; Abke, Jörg (2014): A Retrospective Course Survey of Graduates to Analyse Competencies in Software Engineering. In: IEEE (Hg.): Global Engineering Education Conference (EDUCON): Istanbul, Türkei, S. 100–106. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Geringer Anteil (15%)
Theoretische (pädagogische) Begründung des Ansatzes	Wichtiger Anteil (mind. 20%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Geringer Anteil (15%)

Gold, Carolin; Sedelmaier, Yvonne; Abke, Jörg

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2014): A Multi-Level Didactical Approach to Build up Competencies in Requirements Engineering. In: Birgit Penzenstadler, Sarah Gregory und Dieter Landes (Hg.): 8th International Workshop on Requirements Engineering Education & Training (REET 2014): Karlskrona, Schweden, S. 26–34. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Gleicher Anteil (ca. 50 %)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Gleicher Anteil (ca. 50 %)

Sedelmaier, Yvonne Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2014): A Multi-Perspective Framework for Evaluating Software Engineering Education by Assessing Students' Competencies. SECAT - A Software Engineering Competency Assessment Tool. In: IEEE (Hg.): 44th Frontiers in Education (FIE): Madrid, Spanien, S. 2065–2072. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Überwiegender Anteil (90%)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Überwiegender Anteil (80%)

Sedelmaier, Yvonne Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2014): Software Engineering Body of Skills. In: IEEE (Hg.): Global Engineering Education Conference (EDUCON): Istanbul, Türkei, S. 395–401. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Allein-Autorenschaft (100%)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Allein-Autorenschaft (100%)

Sedelmaier, Yvonne

Landes, Dieter

Unterschriften der AutorInnen

2013

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2013): A Research Agenda for Identifying and Developing Required Competencies in Software Engineering. In: International Journal of Engineering Pedagogy (iJEP) H. 2, S. 30–35. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Überwiegender Anteil (ca. 80 %)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Überwiegender Anteil (ca. 80 %)

Sedelmaier, Yvonne

Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Abke, Jörg; Schwirtlich, Vincent; Sedelmaier, Yvonne (2013): Kompetenzförderung im Software Engineering durch ein mehrstufiges Lehrkonzept im Studiengang Mechatronik. In: Axel Schmolitzky, Detlef Rick und Peter Forbrig (Hg.): HDI 2012 – Informatik für eine nachhaltige Zukunft. 5. Fachtagung zur Hochschuldidaktik der Informatik: Universität Hamburg. Potsdam: Universitäts-Verlag Potsdam (Commentarii informaticae didacticae (CID)), S. 79–84. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Geringer Anteil (ca. 20%)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Überwiegender Anteil (50%)

Abke, Jörg; Schwirtlich, Vincent; Sedelmaier, Yvonne

Unterschriften der AutorInnen

Publikation:

Sedelmaier, Yvonne; Claren, Sascha; Landes, Dieter (2013): Welche Kompetenzen benötigt ein Software Ingenieur? In: Andreas Spillner und Horst Lichter (Hg.): Software Engineering im Unterricht der Hochschulen 2013. 13. Workshop "Software Engineering im Unterricht der Hochschulen" (SEUH 2013): Aachen, S. 117–128. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Überwiegender Anteil (ca. 50 %)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Überwiegender Anteil (ca. 50 %)

Sedelmaier, Yvonne Claren, Sascha; Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Abke, Jörg; Gold, Carolin; Roznawski, Nina; Schwirtlich, Vincent; Sedelmaier, Yvonne (2013): A new approach to collaborative learning in software engineering focussed on embedded systems. In: IEEE (Hg.): International Conference on Interactive Collaborative Learning (ICL). Kazan, Russia, S. 610–616. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Geringer Anteil (ca. 20%)
Theoretische (pädagogische) Begründung des Ansatzes	Überwiegender Anteil (80%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Wichtiger Anteil (30%)

Abke, Jörg;

Gold, Carolin;

Roznawski, Nina;

Schwirtlich, Vincent;

Sedelmaier, Yvonne

Unterschriften der AutorInnen

2012

Publikation:

Sedelmaier, Yvonne; Landes, Dieter (2012): A Research Agenda for Identifying and Developing Required Competencies in Software Engineering. In: IEEE (Hg.): 15th International Conference on Interactive Collaborative Learning (ICL); 41st International Conference on Engineering Pedagogy: Villach, Österreich. (peer reviewed)

Konzeption des beschriebenen Ansatzes	Überwiegender Anteil (ca. 80 %)
Theoretische (pädagogische) Begründung des Ansatzes	Allein-Autorenschaft (100%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Überwiegender Anteil (ca. 80 %)

Sedelmaier, Yvonne

Landes, Dieter

Unterschriften der AutorInnen

Publikation:

Landes, Dieter; Sedelmaier, Yvonne; Pfeiffer, Volkhard; Mottok, Jürgen; Hagel, Georg (2012): Learning and teaching software process models. In: IEEE (Hg.): Global Engineering Education Conference (EDUCON): Marrakesch, Marokko, S. 1153–1160. (peer reviewed) (best-paper-award)

Konzeption des beschriebenen Ansatzes	Geringer Anteil (weniger als 20%)
Theoretische (pädagogische) Begründung des Ansatzes	Wichtiger Anteil (mind. 20%)
Schreiben, Strukturierung und inhaltliche Überarbeitung	Wichtiger Anteil (mind. 20%)

Landes, Dieter; Sedelmaier, Yvonne; Pfeiffer, Volkhard;

Mottok, Jürgen; Hagel, Georg

Unterschriften der AutorInnen