Welche Kompetenzen benötigt ein Software Ingenieur?

Yvonne Sedelmaier, Sascha Claren, Dieter Landes, Hochschule für angewandte Wissenschaften Coburg

{sedelmaier, sascha.claren, landes}@hs-coburg.de

Zusammenfassung

Software ist Teil der modernen Welt und nahezu überall zu finden. Daher gewinnt Software Engineering zunehmend an Bedeutung. Um Software zu entwickeln oder weiterzuentwickeln, sind vielfältige Kompetenzen und Kenntnisse nötig. Das stellt auch die Hochschulausbildung und damit verbunden die didaktische Forschung im Software Engineering vor große Herausforderungen.

In diesem Beitrag wird zunächst das Forschungsinteresse beschrieben und begründet. Dann wird der Kompetenzbegriff diskutiert und erläutert. Nachfolgend werden die methodischen Schritte beschrieben, mit denen das Kompetenzprofil eines Software Ingenieurs erhoben wird. Dabei wird zunächst die Vorgehensweise beschrieben, bevor erste Ergebnisse vorgestellt werden. Zuletzt folgt ein Ausblick, wie die Hochschulausbildung im Software Engineering noch besser auf Kompetenzentwicklung ausgerichtet werden kann.

1. Motivation

Durch die stetig wachsende Komplexität von Software steigen auch die Anforderungen an Informatiker, die mit einer zunehmenden Anzahl von komplexen Rollen, Schnittstellen und Aufgaben konfrontiert werden.

Software wird häufig in großen Teams mit einer Vielzahl fachlicher Rollen entwickelt. Das reicht vom Anforderungsanalysten über den Software Architekten und den Programmierer bis hin zum Software Tester. In jeder dieser Rollen sind zum Teil sehr unterschiedliche Kompetenzen notwendig. Da Software normalerweise nicht für Informatiker, sondern für "fachfremde" Auftraggeber entwickelt wird, muss zum Beispiel derjenige, der Anforderungen erhebt, über ein hohes Maß an interdisziplinärer Kommunikationsfähigkeit verfügen. Ein Software Ingenieur im Bereich Anforderungsanalyse muss in der Lage sein, sich auf für ihn fremde Denkmuster und Denkstrukturen einzulassen, sie zu verstehen und auch zwischen ihnen und der Informatik zu vermitteln. Des Weiteren muss ein Anforderungsanalytiker mit einem gewissen Verständnis für den künftigen Anwendungskontext der zu entwickelnden Software ausgestattet sein, um funktionale und nichtfunktionale Anforderungen möglichst umfassend erheben zu können. Dies setzt ein gewisses Maß an Empathie und Weitblick voraus, sich in die künftige Anwendungssituation und den Anwender hineinzuversetzen. So birgt jede Rolle im Prozess des Software Engineering ihre besonderen Herausforderungen.

Software Ingenieure müssen sich jedoch unabhängig von ihrer Rolle in ein Entwicklungsteam eingliedern können und benötigen neben ihrem fachlichen Wissen auch eine Vielzahl weiterer Kompetenzen. So müssen sie etwa in der Lage sein, Herausforderungen und Probleme rechtzeitig zu erkennen und einzuschätzen, benötigen also ein gewisses Problembewusstsein, um im richtigen Moment angemessen agieren und reagieren zu können. Software Ingenieure müssen ihr erworbenes Wissen ständig auf neue Situationen anpassen und erlernte Methoden selbständig anwenden.

Dieser Artikel konkretisiert und strukturiert den Kompetenzbegriff im Kontext von Software Engineering und stellt ein Modell zur Beschreibung von Kompetenzen vor. Er beleuchtet, wie es entwickelt wurde und welche Anforderungen es erfüllen soll. Er zeigt auf Grundlage des vorgeschlagenen Beschreibungsmodells erste Ergebnisse zu Soll-Kompetenzen eines Software Ingenieurs.

2. Forschungsinteresse

Ein Software Ingenieur benötigt neben einer Vielzahl an fachlichen auch überfachliche Kompetenzen, damit er sein fachliches Wissen in komplexen Situationen überhaupt erfolgreich umsetzen kann. Da es in Lernprozessen keine direkten und klaren Ursache-Wirkungs-Zusammenhänge gibt, können Kompetenzen lediglich trainiert und gefördert, jedoch nicht direkt vermittelt werden. Das stellt die Hochschulausbildung im Software Engineering vor große Herausforderungen. Sie möchte sowohl fachliche als auch überfachliche Kompetenzen im Software Engineering trainieren. Daher starteten sechs bayerische Hochschulen das Projekt EVELIN (Ex-

perimentelle VErbesserung des Lernens von Software EngINeering), um genau hier anzusetzen und das Lehren und Lernen von Software Engineering in seiner gesamten Bandbreite besser verstehen zu können und im Rahmen der Hochschullehre zielgerichtet Rahmenbedingungen zu schaffen, die das Trainieren von Kompetenzen optimal ermöglichen.

Bevor es aber möglich ist, sich darüber Gedanken zu machen, wie man am besten Kompetenzen trainieren kann, stellen sich folgende Fragen:

- Was meint überhaupt "Kompetenz"?
- Wozu dient eine Kompetenzbeschreibung?
- Wie beschreibt man Kompetenzen sinnvoll?
- Was genau kennzeichnet einen guten Software Ingenieur? Welche Kompetenzen soll ein Software Ingenieur haben?

3. Was ist "Kompetenz"?

Der Kompetenzbegriff ist relativ unscharf. Es gibt zahlreiche Definitionen, was unter Kompetenzen verstanden werden kann.

Erpenbeck (2007) schlägt eine Einteilung in personale, aktivitäts- und umsetzungsorientierte, fachlich-methodische sowie sozial-kommunikative Kompetenz vor. Die Definition nach Erpenbeck ermöglicht jedoch keine eindeutige Zuordnung einzelner Kompetenzen zu einer dieser Arten.

Spricht man allgemein von einer Kompetenz, um z.B. eine bestimmte Problemstellung lösen zu können, treten verschiedene Kompetenzarten in Kombination auf. Um etwa Anforderungen erfassen zu können, bedarf es sowohl fachlich-methodischer (z.B. Anforderungen korrekt beschreiben) als auch sozial-kommunikativer Kompetenzen (z.B. Kommunikationsfähigkeit, Teamfähigkeit). Auch Erpenbeck (2007, S. XII) kommt zu dem Schluss, dass Kompetenzen mehr sind als reines Wissen oder Fertigkeiten: "Kompetenzen schließen Fertigkeiten, Wissen und Qualifikationen ein, lassen sich aber nicht darauf reduzieren."

Weinert (2001, S. 27f) gibt eine allgemeinere Definition: "Kompetenzen sind die bei Individuen verfügbaren oder von ihnen erlernbaren kognitiven Fähigkeiten und Fertigkeiten, bestimmte Probleme zu lösen, sowie die damit verbundenen motivationalen (das Motiv betreffend), volitionalen (durch den Willen bestimmt) und sozialen Bereitschaften und Fähigkeiten, die Problemlösungen in variablen Situationen erfolgreich und verantwortungsvoll nutzen zu können." Angelehnt an Weinert unterscheiden wir im vorliegenden Artikel zwei Kompetenzarten:

- Fachkompetenzen entsprechen den kognitiven Fähigkeiten und Fertigkeiten.
- Überfachliche Kompetenzen beinhalten die motivationalen, volitionalen und sozialen Bereitschaften und Fähigkeiten.

Somit beschreiben überfachliche Kompetenzen hier diejenigen Kompetenzen, die gemeinhin nicht direkt als "Fachwissen" bezeichnet werden, sondern vielmehr alle Fähigkeiten, die nötig sind, um Fachwissen situationsbezogen und zielgerichtet in komplexen Situationen anwenden zu können.

Die derzeitige Unterscheidung von lediglich zwei Kompetenzarten ohne weitere Unterteilungen liegt darin begründet, dass aktuell noch keine Aussagen über feinere sinnvolle Differenzierungen getroffen werden können. Das Interesse liegt zunächst auf einer klaren Strukturierung von Forschungsdaten. Erst später ist möglicherweise eine weitere Differenzierung sinnvoll und notwendig.

4. Wozu sollten Kompetenzen beschrieben werden?

Die erforderlichen fachlichen und überfachlichen Kompetenzen im Software Engineering sollen erhoben, analysiert und beschrieben werden, um darauf aufbauend Lehrziele, didaktische Methoden und kompetenzorientierte Prüfungen (weiter) zu entwickeln. Die erhobenen Kompetenzen werden in einem Kompetenzprofil zusammengefasst und stellen so ein umfassendes Bild dar, was ein Software Ingenieur unmittelbar nach der Hochschulausbildung kennen und können soll. Kompetenzprofile sollen sowohl für den Bereich der Kerninformatik als auch für Software Engineering im Nebenfach wie etwa Mechatronik erfasst werden. Darauf aufbauend soll erforscht werden, welche Einflussfaktoren die Kompetenzförderung im Software Engineering begünstigen, so dass in der Hochschulausbildung gezielt kompetenzfördernde Rahmenbedingungen geschaffen werden können.

Die Kompetenzbeschreibung ist somit eine wesentliche Grundlage für die weitere empirisch-experimentelle Didaktikforschung. Kompetenzbeschreibungen werden benötigt, um die Ziele der akademischen Ausbildung klar beschreiben zu können und eine Vergleichsmöglichkeit zu schaffen, inwieweit diese Ziele dann tatsächlich erreicht wurden. Dazu ist eine systematische, vergleichbare Beschreibung von Kompetenzen notwendig. Kompetenzbeschreibungen bilden diese Vergleichsgrundlage, um später in Experimenten zu evaluieren, wie sich die Soll- von den Ist-Kompetenzen Studierender unterscheiden und ob die gezielte Veränderung einzelner Einflussvariablen auf den Lernprozess die Kompetenzentwicklung begünstigt hat. Die Beschreibung von Kompetenzprofilen stellt in diesem Zusammenhang eine Datenbasis zu verschiedenen Zeitpunkten sowie verschiedener Kohorten und Studiengänge dar, die dann miteinander abgeglichen werden können.

Fachliche und überfachliche Kompetenzen unterscheiden sich in ihrer Art, Beschreibung, Förde-

rung und auch Messung stark und werden daher zunächst separat betrachtet. Überfachliche Kompetenzen sind sehr schwer zu fassen, zudem kaum eindeutig definierbar und messbar.

Fachliche Kompetenzen werden mit einem geeigneten Beschreibungsschema erfasst, analysiert und beschrieben. Eine Grundlage dafür bildet SWEBOK (Abran et al. 2004), das die fachlichen Kompetenzen eines Software Ingenieurs mit vier Jahren Berufserfahrung beschreibt. Für überfachliche Kompetenzen im Software Engineering soll in EVELIN ein Schema zur Beschreibung entwickelt und mit konkreten Daten gefüllt werden. Das Ergebnis entspräche dem SWEBOK auf fachlicher Seite. Die erforderlichen überfachlichen Soll-Kompetenzen werden völlig offen und unvoreingenommen ermittelt, um ein möglichst umfassendes Bild zu bekommen. Es geht auch darum zu verstehen, was z.B. "kommunikative Kompetenz" im Kontext von Software Engineering bedeutet.

Die systematische Erfassung der Soll-Kompetenzen ist Voraussetzung dafür, Lernprozesse besser analysieren und verstehen zu können. Zentrale Fragen sind dann: Welche Faktoren wirken überhaupt auf das Lernen? Und wie können diese Einflussfaktoren systematisch erfasst werden, so dass dann zielgerichtet diejenigen angepasst und verändert werden können, die Lernprozesse unterstützen und so die Soll-Kompetenzen gezielt fördern? Hinzu kommt die Schwierigkeit, dass Wechselwirkungen zwischen den Einflussfaktoren bestehen: die Änderung einzelner Faktoren hat immer Auswirkungen auf andere. Es gilt, diese Einflussfaktoren auf Lernprozesse im Software Engineering und ihre Wechselwirkungen untereinander mittels Grounded Theory besser zu verstehen, zumal es speziell für den Bereich Software Engineering noch keine etablierte Theorie gibt.

5. Grounded Theory als Forschungsstrategie

Den methodischen Rahmen für das gesamte Vorhaben bildet die Forschungsstrategie Grounded Theory (Glaser et al. 1998), die häufig verwendet wird, wenn eine auf Daten gestützte Theorie zu einem Thema entwickelt werden soll. Sie zielt darauf ab, Sachverhalte besser zu verstehen und wird in diesem Forschungsprojekt angewandt, um Prozesse umfassend zu verstehen, die dem Lernen von Software Engineering zugrunde liegen. Dieses Verständnis ist Voraussetzung für die kompetenzfördernde Gestaltung der Lernprozesse.

Den Ausgangspunkt der Grounded Theory bilden nicht theoretische Vorannahmen. Vielmehr handelt es sich um einen zirkulären Analyseprozess, der zugleich induktive und deduktive Vorgehenselemente beinhaltet. Ausgangspunkt ist eine erklärende Hypothese, die versucht, von einer Folge auf Vorhergehendes zu schließen (Abduktion). Dann wird ein Typisierungsschema für Hypothesen entwickelt (Deduktion), das mit Forschungsdaten abgeglichen wird (Induktion) (Flick et al. 2000).

Grounded Theory besitzt unter anderem folgende wesentliche Merkmale:

- Eine Theorie wird aus Daten generiert, statt "nur" verifiziert. Der Fokus liegt auf der Entdeckung und Einbeziehung neuer Erkenntnisse, nicht auf der Verifizierung, denn das würde neu auftauchende Perspektiven unterdrücken. Folglich kann eine Theorie nie "fertig" sein, da jederzeit neue Aspekte auftauchen können.
- Ziel ist es, durch das Vergleichen von Gruppen zum einen Kategorien zu bilden und zum anderen bestehende Beziehungen zwischen diesen Kategorien an den Tag zu bringen. Nach Glaser et al. (1998, S. 48f.) "... muss unterstrichen werden, dass die Hypothesen zunächst einen vorläufigen Status haben: Sie beschreiben mutmaßliche, nicht getestete Zusammenhänge zwischen den Kategorien und ihren Eigenschaften - was natürlich nicht heißt, dass man die Hypothesen nicht so gut wie möglich verifizieren sollte. [...] Hypothesen zu generieren heißt, sie im empirischen Material zu verankern - nicht, genug Material anzuhäufen, um einen Beweis führen zu können."
- Grounded Theory fordert eine parallele Datenerhebung und -analyse, und zwar vom Beginn der Forschung an bis zur ihrem Ende.
- Theoretisches Sampling meint, dass die Stichprobe sich während des gesamten Forschungsprozesses weiter entwickelt, so dass bis zum Ende keine endgültigen Aussagen über die erhobenen und einbezogenen Daten getroffen werden können. Die entstehende Theorie steuert, wann im Forschungsprozess welche Daten wo erhoben werden.
- Grundsätzlich gilt ein Mix qualitativer und quantitativer Forschungsmethoden als zielführend.

Daraus resultiert permanente Offenheit für neue Aspekte, die jederzeit in den Forschungsprozess aufgenommen werden.

6. Methodisches Vorgehen

Das Vorgehen der Grounded Theory korrespondiert gut mit dem Anspruch einer angewandten Forschung, da hier sowohl die Forschungsmethodik als auch die Anwendungsorientierung von zu erhebenden Daten ausgehen und diese strukturieren. Zudem soll kein vorgefertigtes Bild verifiziert werden, das es für das Lernen von Software Engineering ohnehin noch nicht gibt. Ein solches Modell könnte nur auf theoretischer Basis entwickelt

werden, hätte dann aber wenig Bezug zur Anwendungsrealität.

Software Engineering ist gekennzeichnet durch ein sehr komplexes Praxisfeld sowie eine Vielzahl zu trainierender Kompetenzen, die zudem von der jeweiligen Rolle abhängen, die ein Software Ingenieur einnimmt. Damit ist der Wissenstransfer von der Theorie auf die praktische Anwendung im Software Engineering eine besondere Herausforderung. Lernen von Software Engineering muss also auf den sehr komplexen Anwendungskontext ausgerichtet sein und wird von zahlreichen Faktoren beeinflusst. Um Anhaltspunkte für Ausgangshypothesen und einen möglichst umfassenden Blick auf diese Einflussfaktoren zu erhalten, ist es sinnvoll, diese Faktoren auf einer detaillierteren Ebene systematisch zu analysieren.

6.1 Überblick über das konkrete Vorgehen

Wird vor diesem Hintergrund die Grounded Theory auf das Lernen von Software Engineering angewandt, ergibt sich folgendes konkretes Vorgehen: Die Kompetenzbeschreibungen ermöglichen einen Vergleich von tatsächlich zu verschiedenen Zeitpunkten bei verschiedenen Studierendengruppen erhobenen Ist- mit den zu erreichenden Soll-Kompetenzen. Auf Grundlage dieser Ergebnisse werden Hypothesen zu den Einflussfaktoren auf das Lernen von Software Engineering gebildet und überprüft. Konkret bedeutet dies, dass zunächst sowohl die Soll-Kompetenzen als auch die Ist-Kompetenzen Studierender erhoben werden.

Dann werden gezielt Einflussvariablen auf den Lernprozess modifiziert und die studentischen Ist-Kompetenzen zu einem späteren Zeitpunkt erneut erhoben. Zuvor definierte Messkriterien zeigen, ob und in welchem Maß sich die Kompetenzen verändert haben, und lassen Rückschlüsse auf die veränderten Einflussvariablen zu. Diese Messkriterien werden über konkrete, auf die Rahmenbedingungen angepasste Lehrziele aus den Soll-Kompetenzen abgeleitet.

Dieser Prozess wird mehrfach durchlaufen, um ein besseres Verständnis über die Einflussfaktoren auf das Lernen von Software Engineering zu erhalten und um so den Lernprozess möglichst kompetenzfördernd gestalten zu können.

Ausgehend von ersten Hypothesen und Daten sollen gezielte Veränderungen und Modifikationen an den Lehr-Lern-Prozessen vorgenommen und dokumentiert werden, so dass Schritt für Schritt bestehende Lehrveranstaltungen weiterentwickelt werden können. Dabei werden - wie bereits beschrieben - Messkriterien festgelegt, anhand derer Kompetenzentwicklung zu verschiedenen Zeitpunkten ablesbar und vergleichbar sein soll. Diese Daten werden dann zu verschiedenen Zeitpunkten erhoben. Aus dem Vergleich der veränderten Einflussvariablen und den Messwerten der Soll- und Ist-Kompetenzen zu verschiedenen Zeitpunkten werden so Rückschlüsse auf mögliche lernfördernde Einflussfaktoren gezogen, die dann gezielt modifiziert werden.

Somit gliedert sich das Vorgehen in zwei wesentliche Schritte: Das Strukturieren und gezielte Verändern von Einflussvariablen auf den Lernprozess und die Kompetenzanalysen zu verschiedenen Zeitpunkten.

Sedelmaier und Landes (2012) beschreiben das Forschungsdesign genauer (vgl. Abb. 4).

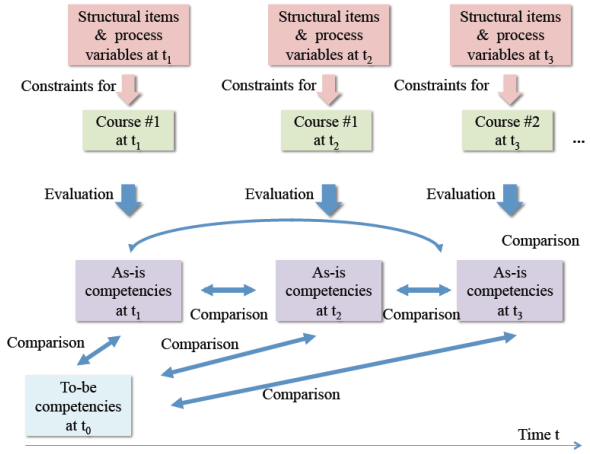

Abb. 4: Überblick über das Forschungsdesign

6.2 Erfassen von Einflussvariablen

Um die Einflussfaktoren auf Lehr-Lern-Prozesse systematisch erfassen zu können, wurden diese zunächst in Struktur-, Prozess- und Ergebnisvariablen (Donabedian 1980) unterteilt. Ergebnisvariablen werden mittels der Kompetenzprofile beschrieben. Strukturvariablen setzen sich aus den didaktischen Aspekten zusammen, die strukturelle Rahmenbedingungen auf den Lernprozess darstellen, wie z.B. Motivationen, Einstellungen, didaktische Kenntnisse der Lehrenden, Räumlichkeiten. Prozessvariablen beeinflussen den tatsächlichen Lernprozess, wie etwa Medien, Lehr- und Lernmethoden.

6.3 Kompetenzanalyse

Um Kompetenzen zu verschiedenen Zeitpunkten erfassen und dokumentieren und Vergleiche anstellen zu können, ist ein einheitliches Beschreibungsschema erforderlich.

Während zu fachlichen Kompetenzen zahlreiche Vorarbeiten zu finden sind, existieren für überfachliche Kompetenzen nur sehr allgemeingültige Einteilungen (vgl. Kap. 3). Daher werden fachliche und überfachliche Kompetenzen separat betrachtet, wodurch sich auch das Vorgehen unterscheidet.

Vorgehen bei überfachlichen Kompetenzen

SWEBOK strukturiert fachliche Inhalte des Software Engineering und bietet somit einen Anhaltspunkt, welche fachlichen Kompetenzen im Software Engineering relevant sein können.

Allerdings vernachlässigt SWEBOK überfachliche Kompetenzen völlig. Zudem ist keine mit SWEBOK vergleichbare "Landkarte" für überfachliche Kompetenzen im Software Engineering bekannt. Auch ist noch unklar, wie trennscharf sich überfachliche Kompetenzen unterscheiden lassen. Lässt sich eine Kompetenz eindeutig beispielsweise den motivationalen oder volitionalen Kompetenzen zuordnen? Diese Frage ist erst nach Erhebung erster konkreter überfachlicher Kompetenzen zu beantworten.

Da im Bereich überfachlicher Kompetenzen für Software Engineering also noch keinerlei Vorerhebungen oder vorstrukturierende Daten vorhanden sind, sind somit die zu erwartenden Inhalte unklar. Daher bietet sich ein iteratives Forschungsdesign an, das bottom-up mit völlig offenem Blick überfachliche Kompetenzen erfasst und dann erst strukturiert. Es wurde ohne ein vorhandenes Schema zur Beschreibung der Kompetenzen begonnen. Dieses entsteht erst parallel mit der Datenerhebung. Die Entwicklung eines Beschreibungsmodells für überfachliche Kompetenzen ist einer ersten Datensammlung nachgelagert.

Überfachliche Soll-Kompetenzen werden aus Sicht unterschiedlicher Stakeholder und zunächst in einem ersten Schritt aus Sicht von Unternehmen für den Bereich Kerninformatik ermittelt. Dazu wurden Interviews, die zu den fachlichen Kompetenzen geführt wurden (siehe unten), nochmals im Hinblick auf überfachliche Kompetenzen ausgewertet. Durch die Nähe zur Datenbasis der Interviews entsteht in diesem bottom-up-Prozess ein Verständnis für die überfachlichen Kompetenzen und ihre Ausprägung. Dieses Vorgehen ermöglicht einen unverstellten Blick auf die tatsächlichen Erfordernisse im Berufsleben und erzeugt schon während des Forschungsprozesses ein Gefühl dafür, was etwa mit "Teamfähigkeit" gemeint ist. Sonst häufig wenig aussagekräftige Worthülsen wie z.B. Teamfähigkeit erfahren durch dieses bottom-up-Vorgehen eine Konkretisierung für den Bereich Software Engineering. Dies ist auch der Grund, warum es in einem ersten Schritt nicht sinnvoll ist, Stakeholder direkt danach zu fragen, welche überfachlichen Kompetenzen benötigt werden. Als Antwort wäre primär eine Aufzählung dieser Worthülsen zu erwarten, es bliebe jedoch offen, was genau im Kontext von Software Engineering darunter zu verstehen ist.

Als erster Schritt zu einem Beschreibungsschema für überfachliche Kompetenzen wurden die vorhandenen Interviews zunächst im Hinblick auf jegliche überfachliche Aussagen codiert. Es entstanden 32 Codes, die unterschiedlich häufig auftraten. Diese Codes deuten auf ein Schema zur Beschreibung überfachlicher Kompetenzen hin, sind jedoch noch weiter zu clustern und zu strukturieren. Gemäß Grounded Theory geschieht dies parallel mit der Erhebung und Auswertung weiterer Daten. In diesem iterativen Prozess entsteht somit neben dem Beschreibungsschema für überfachliche Kompetenzen zeitgleich auch die inhaltliche Beschreibung überfachlicher Kompetenzen, die ein Software Ingenieur benötigt.

Vorgehen bei fachlichen Kompetenzen

Aufgrund anderer Voraussetzungen und Vorkenntnisse wurde bei fachlichen Kompetenzen zuerst top-down ein theoretisches Modell entwickelt und in einem zweiten Schritt mit konkreten Daten, also Kompetenzen, gefüllt und validiert. Das vorgeschlagene Modell zur Beschreibung fachlicher Kompetenzen wurde so in einer Pilotstudie auf seine Verwendbarkeit überprüft. Dazu wurden mittels halbstrukturierter Interviews fachliche Soll-Kompetenzen erhoben und klassifiziert, die Unternehmen von Absolventen im Bereich Kern-Informatik erwarten.

Als Grundlage für mögliche Inhalte und somit für den Interviewleitfaden wurde u.a. SWEBOK herangezogen. Um auch angrenzende Themenkomplexe wie beispielsweise Programmiertechniken und Datenbanken einfließen zu lassen, wurden auch entsprechende fachliche Inhalte vorhandener Lehrveranstaltungen berücksichtigt.

Es wurden Gespräche mit sieben Vertreten von Wirtschaftsunternehmen verschiedener Branchen (vgl. Tabelle 1) im Bereich Software Engineering geführt, aufgezeichnet, transkribiert und ausgewertet. Befragt wurden einerseits Personen mit Entscheidungskompetenz bei der Auswahl neuer Mitarbeiter und Nähe zu den fachlichen Inhalten, also z.B. Geschäftsführer oder Gruppen-/Abteilungsleiter, andererseits Absolventen der eigenen Hochschule mit inzwischen mehrjähriger Berufserfahrung.

Nr	Branche	Entscheider	ehem. Absolvent(in)
1	Marketing, Softwareentwicklung kein Kerngeschäft	X	
2	Softwareentwicklung für Banken und Versicherungen	X	X
3	Bildbearbeitung (Medizin), Embedded, Datenbanklösungen	X	

4	Automobilzulieferer (Embedded)		X
5	Softwareentwicklung und Betrieb einer Marketingplattform	X	
6	Softwareentwicklung und Betrieb einer Marketingplattform		X
7	Großkonzern, Abteilung mit Schwerpunkt Softwareentwicklung	X	

Tabelle 1: Zusammensetzung der Stichprobe

Zwar gewährleistet diese Auswahl noch keine repräsentative Stichprobe, aber immerhin eine relativ gute Abdeckung des fachlichen Spektrums und unterschiedliche Perspektiven. Folgt man der Grounded Theory, genügt dies, um erste Annahmen und Hypothesen zu bilden, die dann den Ausgangspunkt für die weitere Forschung bilden, jedoch noch nicht den Anspruch haben, eine bereits vorhandene These zu validieren.

7. Wie kann man Kompetenzen sinnvoll beschreiben?

Zunächst sind Kompetenzen mittels eines geeigneten Modells zu analysieren und beschreiben. Dazu wurden existierende Klassifikationsschemata auf ihre Eignung im EVELIN-Kontext überprüft.

7.1 Generelle Anforderungen an ein Beschreibungsmodell

Die Untersuchung vorhandener Schemata ergab ein klares Bild der Anforderungen an ein geeignetes Beschreibungsmodell im EVELIN-Kontext. Ein Kompetenzprofil für Software Engineering ist keine rein inhaltliche Sammlung von Stichworten, denn aufgrund der Anwendung in verschiedenen Fachdisziplinen soll es auch ausdrücken können, wie gut Studierende die jeweilige Kompetenz beherrschen. Auch darf ein Kompetenzprofil nicht auf Lehrzielebene formuliert sein, denn hier muss aufgrund der verschiedenen Schwerpunkte einzelner Studiengänge und der Vorstellungen und Priorisierungen der Lehrenden eine gewisse Flexibilität vorhanden sein. Gleichzeitig muss ein Beschreibungssystem auch die Beschreibung, Strukturierung und Einordnung von Lehrzielen unterstützen, so dass es als "Kompass" für die Lehr-Lern-Planung anwendbar ist. Die Lehrziele leiten sich aus dem entsprechenden Kompetenzmodell ab.

Das Modell soll die Beschreibung von Soll- und Ist-Kompetenzen zu verschiedenen Zeitpunkten und deren Vergleich ermöglichen. Da dieses Beschreibungsmodell ein Werkzeug für Planung, Messung und Analyse sein soll, muss es möglichst einfach, aber dennoch ausreichend ausdruckskräftig, verständlich und handhabbar sein.

Wichtig ist auch die Kompatibilität mit anderen Denkansätzen und Beschreibungsmodellen wie z.B. der Taxonomie von Lernzielen im kognitiven Bereich nach Bloom (1972). Es sollte - wenn nötig - zu späteren Zeitpunkten noch erweiterbar sein.

Auch sollte eine Hierarchie aufeinander aufbauender Kompetenzen zwar abbildbar, jedoch nicht zwingend sein.

Klassifikationsschemata für Lernaufgaben und Taxonomien für Software Engineering-Inhalte wurden mit diesen Anforderungen abgeglichen.

7.3 Taxonomien für fachliche Kompetenzen

Die Taxonomie kognitiver Lernziele von Bloom (1972) ist eines der bedeutendsten Klassifizierungssysteme in diesem Bereich. Bloom unterteilt Lernziele in die Hauptklassen Wissen, Verstehen, Anwendung, Analyse, Synthese und Bewertung. Diese sechs Klassen gliedern sich teilweise wiederum in Unterklassen. Insgesamt ergeben sich daraus 24 Klassifikationsmöglichkeiten, was die Einordnung von Kompetenzen erheblich erschwert. Für EVELIN soll jedoch ein Klassifikationsschema eine schnelle und einfache Zuordnung ermöglichen. Trotz der komplexen Unterstruktur ist es schwierig, Kompetenzen verschiedener Bereiche des Software Engineering wie etwa Kerninformatik, Mechatronik oder Wirtschaftsinformatik mit diesem Modell zu unterscheiden. Die Untersuchungen zeigten u.a., dass die Klassifizierung auf der Ebene der Kompetenzen oftmals widersprüchlich ist. Einerseits ist dies auf die strikte kumulative Hierarchie der Klassen zurückzuführen, anderseits sind die Anordnung und Bedeutung der höheren Klassen für diesen Einsatzzweck fraglich (Claren 2012). Da dieses Modell zur Klassifizierung von Lehrzielen entwickelt wurde, eignet es sich nur bedingt für die Beschreibung von Kompetenzen, da diese auf einer abstrakteren Ebene angesiedelt sind.

Anderson und Krathwohl (2001) veröffentlichten eine überarbeitete Fassung der Taxonomie nach Bloom. Gegenüber der Ursprungstaxonomie wurden zunächst die Klassen in aktive Verben umbenannt und die beiden letzten vertauscht, da dies nach Meinung der Autoren der steigenden Komplexität der kognitiven Prozesse besser entspricht. Die wesentliche Änderung ist jedoch die Einführung einer zweiten Dimension, welche die Art des Wissens repräsentiert. Damit ergibt sich das in Abbildung 1 gezeigte zweidimensionale Schema, das zwischen einer kognitiven und einer Wissensdimension unterscheidet.

The Knowledge Dimension	The Cognitive Dimesnion					
	1. Remember	2. Understand	3. Apply	4. Analyze	5. Evaluate	6. Create
A. Factual Knowledge						
B. Conceptual Knowledge						
C. Procedural Knowledge						
D. Metacognitive Knowledge						

Abb. 1: Lernzieltaxonomie nach Anderson und Krathwohl (2001)

Eine weitere Änderung ist der Verzicht auf die kumulativen Eigenschaften der Klassen.

Bei der Anwendung dieser Taxonomie zeigen sich jedoch grundlegende Probleme. Lehrziele bzw. Kompetenzen müssen in diesem Schema immer sowohl einer Wissenskategorie als auch einer Kategorie in der kognitiven Dimension zugeordnet werden. Dabei soll die Formulierung der Lehrziele einen Hinweis für die Einordnung in das Schema geben, was auch die eigentliche Intension des Modells erkennen lässt: Es soll in erster Linie ein Planungs- und Kontrollwerkzeug für den Unterricht sein und geht davon aus, dass bereits formulierte Lehrziele existieren (Anderson und Krathwohl 2001). Im Rahmen von EVELIN sollen Soll-Kompetenzprofile für Software Engineering entstehen, aus denen erst später Lehrziele abgeleitet werden. Generell erhöht die Aufteilung in ein zweidimensionales Schema die Komplexität, da sich für jede zu beschreibende Kompetenz 24 Klassifizierungsmöglichkeiten ergeben. Zu diesem Schluss kommen auch Fuller et al. (2007). Aufgrund der Tatsache, dass höhere Klassen in diesem Modell die darunterliegenden Klassen nicht subsumieren, erlaubt das Modell die Überlappung von Klassen. Ein Lehrziel kann also gleichzeitig in zwei benachbarte Klassen eingeordnet werden. Dadurch fehlt es dem Modell in beiden Dimensionen an der nötigen Trennschärfe, um eine möglichst eindeutige und zielführende Klassifizierung durchzuführen (Claren 2012, Granzer et al. 2008).

Fuller et al. (2007) untersuchten verschiedene Taxonomien zur Beschreibung von Kompetenzen im Bereich der Informatik, darunter auch die Taxonomien nach Bloom sowie nach Anderson und Krathwohl. Ihre Erkenntnisse decken sich weitgehend mit denen der Untersuchung im Rahmen von EVELIN. Darunter fällt etwa die problematische Einordnung in höhere Klassen, die Fragwürdigkeit der hierarchischen Ordnung bei Bloom sowie die zu hohe Komplexität des zweidimensionalen Modells von Anderson (Fuller et al. 2007). Die Autoren schlagen die sogenannte Matrix-Taxonomie (siehe Abbildung 2) vor, die auf der Taxonomie von Anderson basiert.

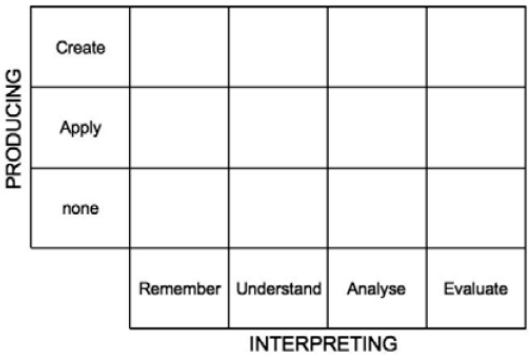

Abb. 2: Matrix-Taxonomie (Fuller et al. 2007)

Die Dimension INTERPRETING umfasst alle Ausprägungen einer Kompetenz, die sich auf Interpretieren und Verstehen beziehen. Die PRODUCING-Ebene dagegen repräsentiert die Fähigkeiten, die nötig sind, um neue Produkte zu entwerfen und entwickeln. Dadurch lassen sich die theoretischen (INTERPRETING) und praktischen (PRODUCING) Aspekte einer Kompetenz beschreiben.

Zunächst wird ein Zielfeld für eine Kompetenz festgelegt, das Studierende über verschiedene Pfade erreichen können. Dabei bewegt sich der Lernprozess vom Startpunkt aus sequenziell entlang der Ebenen zum nächsthöheren Level. Im Lernprozess kann kein Feld übersprungen werden und es ist immer nur eine Bewegung von links nach rechts bzw. von unten nach oben möglich. Im Normalfall beginnt der Lernprozess dabei in REMEBER | NONE (-).

Hier wird deutlich, dass eine andere Zielsetzung als die unseres Projektes vorliegt. Bei Fuller et al. liegt der Fokus auf der Beschreibung, Beobachtung und Planung von Lernpfaden einzelner Studierender oder homogener Studierendengruppen. Während Fuller et al. Prozesse analysieren, beabsichtigt EVELIN, zunächst "Zustände" zu beschreiben.

7.4 Das EVELIN-Modell zur Beschreibung von fachlichen Kompetenzen

Keines der analysierten Modelle eignet sich uneingeschränkt für eine Kompetenzbeschreibung im Rahmen von EVELIN. Für fachliche Kompetenzen wurde daher ein eigenes Modell entwickelt, das bewusst wesentliche Elemente bestehender Taxonomien aufgreift, um sie im Hinblick auf die bereits genannten Anforderungen sinnvoll zu kombinieren und zu erweitern.

Das EVELIN-Modell (vgl. Abbildung 3) besteht aus den Klassen Erinnern, Verstehen, Erklären, Verwenden, Anwenden und (Weiter-)Entwickeln, die sich wie folgt charakterisieren lassen:

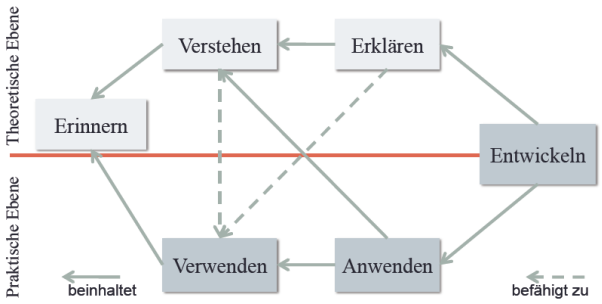

Abb. 3: EVELIN-Modell zur Beschreibung fachlicher Kompetenzen

Erinnern
- sich an Informationen erinnern und sie wortgenau wiedergeben können

Verstehen
- den Sinn / die Bedeutung von Informationen verstehen
- definieren können
- implizites Wissen
- thematisch zugehörige, aber neue Informationen zuordnen können

Erklären
- Zusammenhänge, Abhängigkeiten und Ähnlichkeiten zwischen Informationen erkennen und in eigenen Worten erklären können
- Ursache-Wirkungs-Zusammenhänge theoretisch benennen können
- Informationen strukturieren können
- Vor- und Nachteile erkennen, bewerten und theoretisch analysieren
- Verorten von Informationen in Systemen
- sachliche Analyse
- begründen

Verwenden
- Anwenden in definiertem / eingeschränktem Kontext und / oder unter Anleitung
- Umsetzung in kleinen Schritten oder mittels einer Schablone
- "Kochen nach Rezept"
- Verstehen muss keine Rolle spielen

Anwenden
- selbständiges, eigenverantwortliches Anwenden auch in "schwierigem" Umfeld und / oder Lösungswege situationsgerecht auswählen und umsetzen können
- Verstehen spielt eine Rolle

(Weiter-)Entwickeln
- Nutzen von vorhandenem Wissen zur Entwicklung oder Weiterentwicklung von Lösungen
- Schaffen von neuen Erkenntnissen, Forschung

Dabei sind Erinnern, Verstehen und Erklären auf theoretischer Ebene angesiedelt, während Verwenden, Anwenden und Entwicklung mit praktischer Umsetzung verbunden sind.

8. Welche fachlichen Kompetenzen benötigt ein Software Ingenieur?

Wie in Abschnitt 6.3. beschrieben, wurde eine Befragung von Unternehmensvertretern durchgeführt. Da die Ergebnisse nur einen Baustein in Form einer ersten Datenerhebung zur Bildung von Hypothesen bei der Erstellung der Soll-Kompetenzprofile darstellen, können die Aussagen zu einzelnen Kompetenzen auch noch nicht zu eindeutigen Ausprägungen verdichtet werden, sondern zeigen Tendenzen auf. Dazu trägt auch bei, dass Unternehmen wegen unterschiedlicher Tätigkeitsschwerpunkte unterschiedliche Vorstellungen von Software Engineering haben. Die Entwicklung von Software für Banken und Versicherungen hat etwa auch Auswirkungen auf die erwarteten Kompetenzen im Bereich Softwaretest. Nachfolgend werden wesentliche Ergebnisse der Untersuchung dargestellt, eine detaillierte Diskussion findet sich in Claren (2012).

8.1 Vorgehensmodelle

Bei Vorgehensmodellen gibt es eine Tendenz hin zu umfangreichen theoretischen Kompetenzen. So erwarten die Befragten sowohl bei agilen als auch bei plan-getriebenen Modellen eher Kompetenzen, die sich mit "Erklären" klassifizieren lassen. Eine gute theoretische Basis ist hier wichtig, da die Vorgehensmodelle meist unternehmensspezifisch angepasst sind und dafür bei Neueinsteigern eine Einarbeitung erforderlich ist. Lediglich bei testgetriebener Entwicklung ist die Ausprägung "Anwendung" stärker vertreten.

8.2 Requirements Engineering

Beim Thema Requirements Engineering erwarten die Befragten ein Verständnis dafür, welche Rollen Lasten- bzw. Pflichtenhefte im Anforderungsprozess einnehmen. Auch wenn die Mehrheit der Befragten keine spezielle Technik zur Formulierung von Anforderungen nutzt, erwartet sie dennoch grundlegendes Verständnis dafür, wie sich gute von schlechten Anforderungen unterscheiden. Absolventen sollten Techniken zur Anforderungserhebung sicher und gezielt anwenden können.

8.3 Modellierung

Im Gebiet der Modellierung werden deutlich anwendungsorientierte Kompetenzen erwartet. Absolventen sollten sowohl im Bereich der Systemmodellierung (z.B. mit der UML) als auch der Datenmodellierung (z.B. ER-Modelle) ausgeprägte praktische Erfahrungen im Studium sammeln. Im Gegensatz dazu spielt das Thema Prozessmodellierung bei den Befragten eine eher untergeordnete Rolle.

8.4 Datenbanken

Auch im Themenkomplex Datenbanken dominieren praktisch ausgerichtete Kompetenzen. Vor allem bei Datenbankentwurf und Datenabfrage erwarten die meisten Befragten Kompetenzen auf dem Niveau "Anwenden". Auch bei der Datenbankprogrammierung, wie z. B. dem Erstellen von Triggern und Funktionen, gibt es eine leichte Tendenz in diese Richtung, wobei es manchen Interviewten hier auch genügt, wenn Absolventen "verstehen".

8.5 Design und Implementierung

Hier lassen sich bei zwei Kompetenzen Aussagen über die erwarteten Ausprägungen treffen. Bei der Beherrschung objektorientierter Programmiertechniken sind sich die Befragten überwiegend einig, dass sich ein Absolvent auf Anwendungsniveau befinden sollte. Bei Design-Patterns werden keine praktischen Kompetenzen erwartet. Berufseinsteiger sollten nach Meinung der Befragten begreifen, was hinter diesem Begriff steckt, und die Prinzipien grundlegender Entwurfsmuster verstehen.

8.6 Softwaretest

Bei Kompetenzen im Bereich Softwaretest sind die Meinungen sehr unterschiedlich. Lediglich bei "Unittest" und "Testfallentwurf" können die Mehrzahl der Antworten mit "Anwenden" klassifiziert werden. Außerdem wird erwartet, dass Studienabsolventen die grundlegenden Eigenschaften, Unterschiede und Zusammenhänge verschiedener Testmetriken verstanden haben.

8.7 Konfigurationsmanagement

Die Erwartungen im Bereich Konfigurationsmanagement lassen sich auf rein theoretische Kompetenzen reduzieren. Bei den Themen Change Management, Build Management / Continuous Integration sowie Versionskontrollmechanismen können alle Antworten den theoretischen Klassen "Erinnern", "Verstehen" und "Erklären" zugeordnet werden. Die Erwartungen bei Build Management sind am geringsten, da sich Absolventen nur an grundlegende Aspekte "erinnern" sollen. Bei Change Management oder Versionskontrolle variieren die Antworten stark, jedoch jeweils mit leichter Tendenz zu "Erklären".

8.8 Projektmanagement

Projektmanagement ist überwiegend durch überfachliche Kompetenzen geprägt (z.B. Kommunikations- und Teamfähigkeit). Fachliche Kompetenzen spielen zunächst kaum eine Rolle, da es eher unüblich ist, dass ein Absolvent sofort ein Projekt leitet.

9. Welche überfachlichen Kompetenzen benötigt ein Software Ingenieur?

Wie bereits erläutert kann derzeit noch kein exaktes Schema angegeben werden, das beschreibt, in welcher Ausprägung welche überfachlichen Kompetenzen für Software Engineering konkret erforderlich sind. Es lässt sich jedoch ein erster Eindruck vermitteln, welche Kompetenzen konkret für den Bereich Software Engineering benötigt werden und was darunter zu verstehen ist. Derzeit handelt es sich um die Sichtweise von Software Ingenieuren und Führungskräften. Es geht im Folgenden nicht darum, die genannten Kompetenzen quantitativ aufzuzählen, sondern vielmehr zu verstehen, was zentrale Anforderungen an einen Software Ingenieur sind und was damit gemeint ist.

Dazu wurden in den Interviews insgesamt 306 Stellen codiert, die Rückschlüsse auf überfachliche Kompetenzen zulassen. Insgesamt wurden 32 Codes vergeben, wobei sich über 70 % der codierten Stellen auf die 12 am häufigsten genannten Codes verteilen (vgl. Tabelle 2).

Code	Anzahl Codings	Anteil an allen in %	Dokumente
Problembewusstsein	36	11,43	6
Verstehen komplexer Prozesse	30	9,52	6
Zusammenarbeit	28	8,89	7
Kommunikation mit anderen (auch interdisziplinär)	18	5,71	5
Lernbereitschaft (Gegenteil: Selbstüberschätzung)	18	5,71	5
Problemlösungsfähigkeit/ Kreativität	16	5,08	4
Selbst etwas erarbeiten können	16	5,08	6
Empathie	15	4,76	6
Offenheit für Neues	14	4,44	4
Einfügen in Organisationsstrukturen	13	4,13	5
Praktische Anwendung	12	3,81	3
Abstraktionsvermögen	10	3,17	4

Tabelle 2: Übersicht über die 12 häufigsten Codings für überfachliche Kompetenzen

Unternehmen erwarten also fachlich gut ausgebildete Absolventen, die in der Lage sind, komplexe Prozesse in Unternehmen zu verstehen. Mitarbeiter im Bereich Software Engineering sollen ihre eigene Fachlichkeit realistisch einschätzen. Sie sollen zwar in der Lage sein, sich mit ihrem Wissen im Unternehmen zu verorten, jedoch auch die Bereitschaft mitbringen, sich in Aufbau- bzw. Ablauforganisationen einzufügen. Gleichzeitig wird eine gewisse Aktivität und Selbständigkeit vorausgesetzt. Man erwartet, keine genauen Vorgaben machen zu müssen, wie und wann etwas zu tun ist. Vielmehr soll ein Software Ingenieur sich innerhalb eines Korridors aktiv und eigenverantwortlich bewegen und seine Kompetenzen selbständig einbringen und auch weiterentwickeln. Ebenso notwendig ist die Bereitschaft, sich ständig auf Neues und Unvorhergesehenes einzulassen. Es wird erwartet, dass Absolventen sich selbst und ihre Position im Unternehmen realistisch einschätzen können und so im Team und mit ihren Vorgesetzten konstruktiv zusammenarbeiten und dabei ihren Platz und ihre Rolle finden. Dies ist eng verknüpft mit drei zentralen Anforderungen, die am häufigsten in den Interviews genannt wurden: Problembewusstsein und das Verstehen komplexer Prozesse gepaart mit dem Faktor Zusammenarbeit/ Kommunikation. Zu einem ähnlichen Ergebnis kommen auch Böttcher et al. (2011).

Als zentrale Anforderung wird von einem Software Ingenieur ein ausgeprägtes Problembewusstsein erwartet. Problembewusstsein meint, sich in einem komplexen Prozess zurechtzufinden und zu erkennen, wann welche Kompetenzen wie benötigt werden, und wann und wie sie zusammenhängen. Ein Interviewpartner beschreibt dies folgendermaßen: "... das ist auch so ein Thema, das wurde auch in meinem Studium behandelt, das kann man aber auch erst sozusagen verstehen und adaptieren, wenn man das Problembewusstsein hat, [...] weil man vielleicht schon das eine oder andere Problem wirklich erlebt hat, dass man sagt: o.k., ich habe jetzt eine gewisse Problemstellung und die muss ich irgendwie lösen und dann eben auch dieses Aha-Erlebnis gehabt hat [...]"

Ein anderer Interviewpartner formuliert die Situation so: "Das Problem ist ja, man hört im Studium viele Sachen, die einem erzählt werden und – da heißt es dann: o.k., das macht man so und das macht man so – aber man weiß im Grunde genommen ganz oft nicht, warum macht man das denn eigentlich so. Weil es gibt eigentlich zehn Möglichkeiten, etwas zu machen. Im Studium werden vielleicht zwei Möglichkeiten beleuchtet [...] und dann hat man zwar gehört: Okay, das und das sollte man so machen, aber man weiß gar nicht, warum man das so machen muss und wenn jetzt jemand zu uns kommt, der sich eben über das Studium hinaus mit den Themen beschäftigt hat, dann ist es ganz oft auch der Fall, dass der eben – ja verschiedene Möglichkeiten schon abgeklopft hat und dann auch – ja von sich aus verstehen kann, nachvollziehen kann warum manche Sachen notwendig sind." Auch ein dritter Befragter verlangt Verständnis dafür "...vor allem WARUM muss ich das verwenden, welchen Nutzen ziehe ich denn daraus, wenn ich das mache."

Dabei geht es nicht ausschließlich um das Verstehen und Einordnen von Fach- und Faktenwissen, sondern auch darum, die Tragweite und Zusammenhänge komplexer Probleme bewusst wahrzunehmen. Ein Interviewpartner fasst diese Aspekte so zusammen: "Wenn man dann ins alltägliche Berufsleben hinein geschmissen wird, hat man eher das Problem, dass man von den ganzen… ja… Dingen, die da existierten, erst mal überwältigt ist."

Diese Zitate verdeutlichen, dass der Theorie-Praxis-Transfer von Kompetenzen und Wissen eine zentrale Herausforderung darstellt, die offensichtlich für viele Absolventen nicht leicht zu meistern ist und auch die Unternehmen vor einige Probleme stellt. Umgekehrt ist es für die Hochschulausbildung schwierig, den komplexen Praxisalltag in der Ausbildung abzubilden, so dass die zitierten Aha-Erlebnisse für die Studierenden erfahrbar werden.

Eine weitere wichtige überfachliche Kompetenz, die von fast allen Befragten genannt wurde, ist das Verstehen komplexer Prozesse. Für das Einfinden und angemessene Bewegen in diesen komplexen Abläufen, die dem Software Engineering inhärent sind, ist es absolut notwendig, dass ein Software Ingenieur diese überblickt und versteht. Dies schließt auch "einen Weitblick über seinen eigenen Tellerrand hinaus" mit ein. Ein Befragter beschreibt dies so: "Aber das Prinzip muss er begreifen: wie spielt das alles zusammen."

Ein dritter, sehr wichtiger Komplex, auf den sich überfachliche Kompetenzen beziehen, ist der Bereich der Zusammenarbeit. Einer der Befragten bezeichnet das sehr treffend als "Teamwork-Alltag". Ein anderer Befragter gibt hier einen Einblick in den Arbeitsablauf: "Dieser kommunikative Austausch über Themen nicht nur im Daily Scrum mal ein paar Minuten, sondern eigentlich fachlich und konkret den ganzen Tag über, im Pairing, in immer wiederkehrenden Beratungssituationen. Wir haben auch die Teams grundsätzlich immer zusammensitzen in Räumen oder an Tischinseln von Leuten, die im Team gemeinsam miteinander arbeiten. Dass also auch ein Austausch stattfindet, dass sie nicht nur sich von einem Zimmer ins andere eine E-Mail schreiben, sondern dass sie auch immer diesen mündlichen Kommunikationsweg und dass viele darüber sprechen - das versuchen wir eigentlich in jeder Weise zu befördern." Insgesamt scheint der Faktor Zusammenarbeit zunehmend an Bedeu-

tung zu gewinnen, allerdings verändert sich die Arbeitsrealität in gleichem Maße. Ein Befragter fasst es zusammen: "Teamfähigkeit ist allerdings immer besser ausgeprägt heute, weil eben immer mehr in der Ausbildung – Arbeiten im Team erledigt werden oder erledigt werden müssen. Das hat sich deutlich verbessert im Lauf der 25 oder 30 Jahre, die ich das jetzt betrachte. Am Anfang waren viele, viele, viele Einzelkämpfer da, das hat sich deutlich verbessert und das ist auch gut, die Entwicklung ist gut." Auf der anderen Seite erkennen die Befragten noch immer Handlungsbedarf: "Ich sage mal so, die Leute haben scheinbar eine gewisse technische Kompetenz, aber wie man eine vernünftige Präsentation macht, wie man einen vernünftigen Text schreibt, wie man in einem Team umgeht, wie man Konflikte löst, das sind alles Themen, die hat der gewöhnliche Hochschulabgänger nicht gelernt." Ein weiterer sieht darin einen Ausweg für die Hochschulausbildung, "...dass man Projekte definiert, wo wirklich mal drei, vier, fünf Leute dabei sind. Wo man sich untereinander abstimmen muss. Wo die Aufgaben auch so abgegrenzt sind, dass man notgedrungen Schnittstellen zu anderen hat." Ein weiterer Befragter würde es auch gut finden, wenn den Studierenden auch Projektverantwortung für ein Studienprojekt übergeben würde.

10. Fazit und Ausblick

Insgesamt lässt sich festhalten, dass ein Software Ingenieur eine Vielzahl fachlicher und überfachlicher Kompetenzen benötigt. Nun stellt sich die Frage, wie man diese benötigten Kompetenzen in der Hochschulausbildung am besten fördern kann. Auf diesem Weg ist die Beschreibung der Soll-Kompetenzen mittels eines Schemas, das diese zu verschiedenen Zeitpunkten vergleichbar macht, nur ein erster Schritt.

Die Analyse und Beschreibung der Soll-Kompetenzen für Software Engineering muss noch ausgebaut werden. Dies ist für sowohl für Kerninformatik als auch für alle anderen beteiligten Bereiche wie Mechatronik oder Studiengänge mit nicht primärem Informatikbezug notwendig. Um die Tragfähigkeit der erhobenen Kompetenzprofile weiter zu erhöhen, werden auch die Ergebnisse anderen beteiligten Verbundhochschulen berücksichtigt. Neben der Ausweitung der Datenbasis ist es zudem notwendig, weitere Blickwinkel neben der unternehmerischen Sicht einzubinden. Sinnvoll ist es, beispielsweise auch Absolventen, Studierende oder Personalverantwortliche mit einzubeziehen. Auch kann es sinnvoll sein, die entstandenen Kompetenzprofile für die verschiedenen Bereiche wie Kerninformatik oder Mechatronik miteinander zu vergleichen.

Besonders für überfachliche Kompetenzen gilt es, ein Beschreibungsschema zu entwickeln, das nahezu die gleichen Anforderungen erfüllt wie das für fachliche Kompetenzen. Dieses Modell entsteht parallel zur Analyse der vorhandenen Interviewdaten mit Fokus auf überfachlichen Kompetenzen. Dabei sollen überfachliche Kompetenzen sowohl strukturiert als auch definiert werden. Es gilt, die Frage zu beantworten, was im Kontext von Software Engineering etwa unter Team- oder Kommunikationsfähigkeit verstanden wird. Eine weitere interessante Fragestellung ist, ob überhaupt und ggf. welche Korrelationen zwischen fachlichen und überfachlichen Kompetenzen auftreten. Werden einzelne überfachliche Kompetenzen besonders häufig im Zusammenhang mit einer oder mehreren bestimmten fachlichen Kompetenzen genannt? Für beide Schemata und ihren Inhalt gilt derselbe iterative Prozess, in dem die Daten auch während des gesamten Forschungsprojektes EVELIN immer wieder überprüft und weiterentwickelt werden. Gemäß Grounded Theory fließen ständig neue Erkenntnisse und Aspekte ein, so dass ein möglichst umfassendes Bild zum Lehren und Lernen von Software Engineering entstehen kann.

Parallel dazu findet eine weitere didaktische und inhaltliche Analyse der aktuellen Lehr-Lern-Konzepte für Software Engineering und der Einflussvariablen auf die Lernprozesse statt. Die fachlichen Inhalte von Lehrveranstaltungen werden wiederum in die Weiterentwicklung von Soll-Kompetenzprofilen einfließen. Zudem ist bei der Entwicklung didaktischer Konzepte eine detaillierte Berücksichtigung der zu vermittelnden Inhalte unumgänglich.

Sobald konkretere Soll-Kompetenzprofile vorliegen, ist die Sichtweise der Lehrenden stärker einzubinden. Die Lehrenden nehmen im Lernprozess der Studierenden eine Schlüsselposition ein und prägen besonders auch durch ihre Vorstellungen und "Definitionen gelungenen Lernens" (Bender 2009) das Lehren und Lernen maßgeblich. Daher ist es sinnvoll, diese Sichtweisen zunächst separat mit der erforderlichen Sorgfalt zu berücksichtigen. So sollen nicht nur inhaltlich-fachliche Lehrinhalte mit in die Bildung von Soll-Kompetenzprofilen einfließen, sondern auch Erwartungen, Motive, Werthaltungen, Menschenbild und personenbezogene Merkmale etc. Welche Vorstellungen, Ideale, Ziele und Wertvorstellungen haben die Lehrenden? Darauf aufbauend werden zusammen mit den Lehrenden aus den Soll-Kompetenzprofilen konkrete kompetenzorientierte Lehrziele abgeleitet und Messkriterien für deren Erreichung definiert. Auch dieses Vorgehen ist wiederum ausgerichtet auf die jeweiligen Lehrenden und die Lehrveranstaltungen, denn Lehrziele können nicht allgemeingültig formuliert sein.

Weitere zu erhebende Daten sind die Erwartungen, Motivationen und Vorkenntnisse der Studierenden. In diesem Kontext stellt sich unter anderem die Frage, ob Studierende, die sich für ein bestimmtes Fach entscheiden, signifikante Eigenschaften aufweisen. Auch das sind Faktoren, auf die Lehr-Lern-Konzepte zugeschnitten werden müssen. Können sowohl bei den Lehrenden als auch bei den Studierenden gewisse Systematiken oder Muster erkannt werden, die bei der Entwicklung didaktischer Konzepte berücksichtigt werden müssen und zumindest für Software Engineering oder Teilbereiche davon signifikant auftreten?

So soll versucht werden, die Lücke zwischen den Soll- und Ist-Kompetenzen der Studierenden Schritt für Schritt zu verringern und die Hochschulausbildung im Software Engineering noch weiter zu verbessern.

Danksagung

Wir danken den Gutachtern für hilfreiche Anmerkungen zu einer früheren Version dieses Beitrags.

Das Projekt EVELIN wird gefördert durch das Bundesministerium für Bildung und Forschung unter der Fördernummer 01PL12022A.

Literatur

Abran, A. / Moore, J.W., Hrsg. (2004): Guide to the Software Engineering Book of Knowledge. Los Alamitos, CA: IEEE Computer Society Press. Verfügbar unter: http://www.computer.org/portal/web/swebok/htmlformat

Anderson, L. / Krathwohl, D., Hrsg. (2001): A taxonomy for learning, teaching, and assessing. A revision of Bloom's taxonomy of educational objectives. New York: Longman.

Bender, W.: Wie kann Qualitätsentwicklung aus dem Pädagogischen heraus entwickelt werden? - Pädagogische Reflexivität. In: Dehn, C., Hrsg. (2009): Pädagogische Qualität. Hannover: Expressum-Verlag, 69-77.

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

Böttcher, A., Thurner, V., Müller, G. (2011): Kompetenzorientierte Lehre im Software Engineering. In: Ludewig, J., Böttcher, A., Hrsg. (2011): SEUH 2011, 33-39.

Claren, S. (2012): Erhebung und Beschreibung von Kompetenzen im Software Engineering. Masterarbeit, Hochschule Coburg.

Donabedian, A. (1980): Explorations in Quality Assessment and Monitoring: The definition of quality and approaches to its assessment. Ann. Arbor, MI: Health Administration Press, 1980.

Erpenbeck, J., Hrsg. (2007): Handbuch Kompetenzmessung. Erkennen, verstehen und bewerten von Kompetenzen in der betrieblichen, pädagogischen und psychologischen Praxis. 2. Aufl. Stuttgart: Schäffer-Poeschel.

Flick, U. / von Kardorff, E. / Steinke, I. (2000): Was ist qualitative Forschung? Einleitung und Überblick. In: Qualitative Forschung. Ein Handbuch. Reinbeck: Rowohlt, 13-29.

Fuller, U. et al (2007): Developing a computer sicience-specific Learning Taxonomy. Verfügbar unter http://www.cs.kent.ac.uk/pubs/2007/2798/content.pdf, abgerufen am 24.09.2012.

Granzer, D. / Köller, O. (2008): Kompetenzmodelle und Aufgabenentwicklung für die standardisierte Leistungsmessung im Fach Deutsch. In: Bremerich-Vos, A. / Granzer, D. / Köller, O., Hrsg. (2008): Lernstandsbestimmung im Fach Deutsch. Gute Aufgaben für den Unterricht. Weinheim: Beltz, 10-49.

Glaser, B. / Strauß, A. (1998): Grounded theory - Strategien qualitativer Forschung. Bern: Hans Huber.

Sedelmaier, Y. / Landes, D. (2012): A research agenda for identifying and developing required competencies in software engineering. Proc Int. Conference on Interactive Collaborative Learning 2012.

Weinert, F. (2001): Leistungsmessung in Schulen. 2. Auflage Weinheim: Beltz.