# Vorbereitung auf das wahre Leben — Herausforderungen bei Diskurs und Umgang mit Unsicherheit in der Lehre

Axel Böttcher[1], Daniela Zehetmeier[2]

**Abstract:** Es ist keine Seltenheit, dass unsere Studierenden in ihrem späteren Berufsleben mit Software konfrontiert werden, die über lange Zeit gewachsen ist. Um später die damit verbundenen Herausforderungen meistern zu können, benötigen die Studierenden einerseits die Fähigkeit, mit Unsicherheit umzugehen. Denn Anforderungen sind oft unklar, oder es nicht immer möglich, unzureichend dokumentierte Altsysteme in kurzer Zeit soweit zu verstehen, dass Wartungsarbeiten oder Erweiterungen daran vorgenommen werden können. Andererseits bedarf es der Beurteilungsfähigkeit, um technische Realisierungen zu bewerten und kritische Diskurse zu Software mit Anderen führen zu können.

In diesem Beitrag beschreiben wir am Beispiel des Moduls Software-Archäologie unsere Erfahrungen, diese wichtigen Kompetenzen in der Lehre zu adressieren. Die Beobachtungen zeigen, dass für viele Studierenden das Führen von Diskursen ungewohnt ist und dass sie mit Unsicherheiten schlecht umgehen können. Wir schlussfolgern, dass weitere Methoden entwickelt werden müssen, um diese Kompetenzen zu lehren und zu messen.

**Keywords:** Höhere kognitive Fähigkeiten; Beurteilungsfähigkeit; Diskurs in der Lehre; Software Maintenance and Evolution; Nachhaltige Software

## 1 Motivation

Die Ausbildung unserer Studierenden in Softwareentwicklung und Software Engineering erfordert die Auseinandersetzung mit sehr vielen abstrakten Konzepten und formalen Sprachkonstrukten, die nahezu beliebig miteinander kombinierbar sind. Dies bringt uns in das klassische Dilemma von zu viel Stoff und zu wenig Zeit [Le11]. Wir haben beobachtet, dass wir deshalb nicht nur im Grundstudium dazu neigen, den Fokus der Lehre stark auf die Kompetenzebenen des Verstehens und Anwendens zu setzen.

Wir formulieren für die Lehrveranstaltungen immer wieder Lernziele auf höheren Kompetenzebenen gemäß der überarbeiteten Lernzieltaxonomie von Bloom [An01, Th15] und adressieren diese in Lehre und Prüfung [TBS16]. Dennoch bleiben aus unserer Sicht wichtige Entwicklungsschritte, die mit höheren Kompetenzebenen in Verbindung stehen, außen vor.

[1] Hochschule München, Fakultät für Informatik und Mathematik, Lothstraße 64, D-80335 München, axel.boettcher@hm.edu
[2] Lufthansa Aviation Training GmbH, Südallee 15, D-85356 München-Flughafen, daniela.zehetmeier@lat.dlh.de

doi:10.18420/SEUH2022_06

Ein Beispiel einer solchen vernachlässigten Kompetenz ist die Beurteilungsfähigkeit. In der Praxis ist es essentiell, beurteilen zu können, welche der vielen Möglichkeiten, ein programmiertechnisches Problem zu lösen, sinnvoll sind und welche weniger sinnvoll. Wir unternahmen erste Versuche, diese Kompetenz bereits im Grundlagenmodul Softwareentwicklung zu fördern und in der Prüfung zu messen. Die zur Messung der Kompetenz entwickelte Prüfungsaufgabe erzielte mit einem Mittelwert von 23% der bei dieser Aufgabe erreichbaren Punkte den traurigen Rekord innerhalb dieser Prüfung. Diese Erfahrungen führten uns zu der Hypothese, dass wir in unseren Lehrveranstaltungen dem gemeinsamen Reflektieren der Entwicklungsprozesse und der Evaluation der daraus resultierenden Artefakte zu wenig Raum geben.

Des Weiteren ist die Fähigkeit, mit fremden („Legacy“) Projekten umzugehen, eine wichtige Kompetenz, die wir kaum adressieren. Viele unserer Lehrbeispiele und Praktikumsaufgaben in den ersten Semestern basieren auf glasklaren Anforderungen, um gerade keine Unklarheiten und Unsicherheiten aufkommen zu lassen. Das resultiert aus der Notwendigkeit, bei den großen Anfängerkohorten die Korrekturen und Bewertung effizient durchführen zu können. Außerdem versuchen wir, uns bei der Bewertung nicht angreifbar zu machen, indem wir den Raum für Diskussionen möglichst klein halten. Es ist aber keine Seltenheit, dass Software-Systeme lang leben und über Jahrzehnte hinweg genutzt werden [Du12]. Damit ist es also nicht unwahrscheinlich, dass unsere Studierenden in ihrem späteren Berufsleben mit der Aufgabe konfrontiert werden, Wartungsarbeiten an existierender Software vorzunehmen. Die Arbeit in einem derartigen, nicht mehr engmaschig qualitätsgesicherten Umfeld, bringt für Entwicklerinnen und Entwickler Unsicherheiten mit sich, mit denen sie umzugehen lernen müssen. Neben Erfahrung im Umgang mit Legacy-Projekten und im Lesen fremden Codes hilft dabei zielgerichtete Analyse und Diskussion.

Es mangelt also letztlich am Führen von kritischen Diskursen mit fundierten Argumenten im Hörsaal [HP18]. Wichtig hierbei ist es, dass die Studierenden den Diskurs als gewinnbringend für sich selbst empfinden, und einen Erkenntnisgewinn davon tragen können.

## 2 Zielsetzung

Aufgrund dieser Erkenntnisse wollen wir die Lehre anpassen und die genannten Defizite gezielt in vertiefenden Kursen adressieren. Die oben genannten Punkte lassen sich beispielsweise dadurch in die Lehre integrieren, dass wir die Studierenden mit einer umfangreichen existierenden Codebasis arbeiten lassen. Je weniger ein solches Projekt unter Einhaltung der klassischen Qualitätsstandards entwickelt wurde, desto mehr müssen wir dabei, wie die Archäologen es formulieren würden, „Erkenntnisse aus dem ziehen, was im Altertum von Menschenhand geschaffen worden ist“ [Si00]. Ausgehend von einer solchen metaphorischen Anspielung an die Altertumswissenschaften wurde bereits 2002 von Hunt und Thomas der Begriff *Software-Archäologie* vorgeschlagen [HT02].

In diesem Beitrag beschreiben wir am Beispiel eines Moduls Software-Archäologie unsere Erfahrungen damit, diese wichtigen Themen im Bereich des Software Engineering bzw. der Software Maintenance bzw. Evolution in der Lehre zu adressieren, die aus unserer Sicht oft außen vor bleiben. Es soll ein Rahmen geschaffen werden, für gemeinsame Diskurse auf Augenhöhe mit den Studierenden, zum Inhalt und zur Qualität fremder Software.

Übergeordnetes Ziel des Moduls ist es auch, die Studierenden für die Wertschätzung von bestehender Software zu sensibilisieren. Beispielsweise ist die Auswahl einer Programmiersprache und damit verbundener Frameworks und Teststrategien immer im Kontext des Entwicklungszeitpunkts zu sehen. Berücksichtigt werden müssen immer auch Tatsachen wie etwa, dass viele Programmierer:innen Quereinsteiger aus unterschiedlichen Bereichen waren, wie der Elektrotechnik oder Physik. Und dass damit Anwendung von Prinzipien wie Modularisierung, Separation of Concerns etc. oft vernachlässigt wurden.

## 3 Literatur

Software Maintenance wird durchaus in Standard-Curricula erwähnt. Die GI-Empfehlungen für Bachelor- und Masterprogramme der Informatik [Zu16] erwähnen Folgendes als Teil der Realisierungskompetenz: „Für die Wartung und Erweiterung von Software ist die Fähigkeit notwendig, sich in vorhandenen Quelltext einzuarbeiten und diesen sinnvoll weiter zu entwickeln“.

Software-Wartung als Teildisziplin des Software Engineering wird selten als eigenes Modul, wenn überhaupt, dann als ein Bestandteil im Rahmen einer Lehrveranstaltung zu Software Engineering unterrichtet.

Zur Vernachlässigung des Lesens fremden Quellcodes stellten bereits 2003 van Deursen et al. in einem Workshop-Beitrag für *Program Comprehension* fest: „Students learn how to write new programs but they are not taught how to read and change existing and large ones“ [De03].

Auf der didaktischen Seite analysiert Bower Hörsaal-Diskurse mit Methoden der strukturierten Inhaltsanalyse [Bo09]. Bower fasst in seiner Arbeit jedes Zwiegespräch in der Lehre zur Softwareentwicklung als Diskurs auf. Wir verstehen hier unter Diskurs nur die kritische Auseinandersetzung mit den Artefakten und ihren Entstehungsprozessen.

Smith wie auch Pinto verfolgen den Ansatz, Maintenance und Evolution am existierenden Open-Source-Projekt zu unterrichten [Sm14, Pi17]. Die Bedeutung von kritischem Diskurs wird dabei nicht erwähnt. Wir erweitern dieses Feld zur Software-Archäologie um eine retrospektive Architektur- und Anforderungsanalyse. Dabei fokussieren wir in der Lehre auch die kritische Diskussion der Artefakte und die Beurteilung der Analysemethoden.

# 4 Kursdesign

Das Modul Software-Archäologie ist als Wahlfach mit fünf ECTS für die Bachelor-Studiengänge Informatik und Wirtschaftsinformatik konzipiert. Es ist für je zwei Semesterwochenstunden seminaristischem Unterricht sowie Praktikum konzipiert. Als Prüfungsform ist eine zweigeteilte Prüfung, bestehend aus einer benoteten Studienarbeit sowie einer benoteten mündlichen Prüfung vorgesehen.

Corona-bedingt musste der Kurs im Sommersemester 2021 komplett online stattfinden. Wir hatten die Möglichkeit das Modul im Pair-Teaching zu unterrichten [ZBBK18]. Dieses Setting ermöglichte das Vorleben von Diskursen durch Diskussionen zwischen den Dozierenden.

## 4.1 Lernziele des Moduls

Folgende Lernziele formulierten wir ausgehend von den Kompetenzen, die aus unserer Sicht für die gegebenen Aufgaben erforderlich sind.

- Sie analysieren existierenden Quelltext, um ihn zu verstehen, und um
  - Rückschlüsse auf die Intention der ursprünglichen Entwickler:innen zu ziehen
  - Requirements zu identifizieren, sodass diese als Grundlage für Refactorings oder eine Re-Implementierung dienen können
- Sie dokumentieren die gewonnen Erkenntnisse mit geeigneten Mitteln.
- Sie wenden Techniken des Reverse Engineering systematisch und gezielt an.
- Sie wenden Techniken des Refactoring systematisch und gezielt an.
- Sie analysieren Kontrollfluss theoretisch oder anhand existierender Ausgangsdaten.
- Sie entwerfen und implementieren Testinfrastruktur für Legacy Code und führen diese aus.
- Sie diskutieren Vorgehensweisen und Arbeitsergebnisse in Ihrer Praktikumsgruppe und im Plenum.

## 4.2 Projekt im Modul

Im Zentrum der Lehrveranstaltung sollte ein existierendes Projekt stehen, an dem entlang sich eine inhaltliche Auseinandersetzung innerhalb eines Semesters orientieren kann. Ähnlich wie Smith et al. [Sm14] formulierten wir vorab Kriterien an ein solches Projekt:

- Das Projekt muss hinreichend groß und komplex sein, sodass keine Gruppe von Studierenden versucht, eine komplette Reimplementierung „am Wochenende“ vorzunehmen.
- Das Projekt muss technische Schulden aufweisen.
- Das Projekt muss verschiedene Schnittstellen bedienen.

- Das Projekt sollte nicht aus einem akademischen Kontext stammen.

Insofern kämen prinzipiell Projekte aus dem Open-Source-Umfeld in Betracht, ebenso wie Projekte aus der Praxis [Pi17]. Anders als z. B. in [Pi17] wollten wir gerade keine Open Source Community, die als Fallback bei Schwierigkeiten an uns vorbei kontaktiert werden hätte können.

Ähnlich wie von Smith et al. [Sm14] erwähnt, hatte auch unsere Suche nach einem geeigneten Projekt viel Zeit verschlungen und war zunächst erfolglos geblieben. Mit dem Wechsel der Autorin in die Industrie eröffnete sich die Möglichkeit, auf mehrere Projekte aus dem beruflichen Kontext zuzugreifen, welche die Anforderungen erfüllen. Die Auswahlentscheidung fiel auf ein historisch gewachsenes, ca. zwölf Jahre altes, in ColdFusion [Na10] realisiertes Projekt. Die Software ist nach wie vor im produktiven Einsatz. Der Umfang beträgt etwa 30.000 Codezeilen. Das System bedient mehrere Schnittstellen, ist weitgehend undokumentiert, verfügt über keinerlei Tests und bringt diverse weitere technische Schulden mit sich.

### 4.3 Themen und Aufgabenstellungen

In den ersten beiden Dritteln des Semesters sollte die bestehende Software analysiert werden mit dem Ziel, eine am arc42-Template [SH16] orientierte Architekturdokumentation zu erstellen. Der letzte Schritt besteht dabei in einer Beschreibung von Anforderungen, die sich ex-post aus der Software extrahieren lassen. Neben einer Unterstützung von Wartungsarbeiten kann die Dokumentation so eine Grundlage für die anstehende Reimplementierung des Systems bieten. Deshalb war dann im letzten Teil ein modernes User Interface auf Basis der Anforderungen zu entwerfen, sodass der Nutzen der archäologischen Arbeit sichtbar wird.

Die thematische Struktur umfasste folgende Themenblöcke:

**Glossar** Die erste Aufgabenstellung umfasste neben der Einrichtung des Projekts das Anlegen eines Glossars. Die Vorgabe war, das Glossar fortlaufend zu erweitern, um die im Laufe der Zeit jeweils hinzugewonnenen Erkenntnisse.

**Extraktion einer API-Dokumentation** Diese Aufgabe verlangte die Erstellung einer Dokumentation der (pre-REST) HTTP-API des Projektes einschließlich einer Beschreibung der gewählten Vorgehensweise in einem Wiki. Die konkrete Gestaltung der Dokumentation war den Studierenden überlassen.

**Schnittstellenbeschreibung und Querschnittsthemen** Das arc42-Template anzulegen und mit Glossar sowie API-Dokumentation zu füllen war der dritte Arbeitsauftrag. Darüber hinaus mussten Stakeholder, Randbedingungen und Kontext identifiziert, sowie querschnittliche Konzepte und externe Schnittstellen des Systems „soweit möglich“ beschrieben werden.

**Dokumentation der Datenbank** In diesem Schritt sollten die Beziehungen zwischen den existierenden Datenbanktabellen durch Reverse Engineering analysiert, beschrieben und evaluiert werden.

**Laufzeitsicht** Nachdem die Studierenden sich ausführlich mit der statischen Sicht auf die Applikation befasst hatten, sollten sie im nächsten Schritt dynamische Sichten erfassen. Einzelne API-Backend-Funktion waren in Aktivitätsdiagramme mit wesentlicher Abstraktion zu überführen.

**Dokumentation der Anforderungen** Eine ex-post-Extraktion der Anforderungen war als Grundlage für eine Reimplementierung der Anwendung gedacht.

**GUI** Am Ende des Semesters sollte die Studierenden das Wissen das sie über die Applikation und den fachlichen Kontext erworben haben in ein Re-Design des User Interface fließen lassen.

Zu jedem Thema wurde ein Aufgabenblatt ausgegeben. Die Aufgaben waren in Vierer-Teams zu bearbeiten. Ergebnisse wurden im Plenum vorgestellt und gemeinsam diskutiert.

## 5 Beobachtungen und Reflexion

Uns war bei Konzeption der Lehrveranstaltung wichtig, dass das Projekt viele Unsicherheiten beinhaltet. Die historisch gewachsene Software konfrontierte auch uns Dozierende mit einer gewissen Unsicherheit, denn der komplette Funktionsumfang der Applikation hatte sich auch uns bis zum Ende der Veranstaltung nicht vollständig erschlossen.

Während der Vorbereitung der einzelnen Lehrveranstaltungsstunden, aber auch in der Retrospektive, diskutierten wir immer wieder verschiedene Aussagen, Arbeitsergebnisse, aber auch Verhaltensweisen von Studierenden. Drei wesentliche Erkenntnisse scheinen uns an dieser Stelle einer besonderen Erwähnung wert: inhaltliche Diskussionen, die sich in der Lehrsituation ergeben, Aufgabenstellungen, die allein durch den nicht vollständig erschlossenen Kontext vage sind, sowie der Umgang mit der daraus resultierenden Unsicherheit.

### 5.1 Diskussionen und Raum für Fragen

Wir Lehrenden sahen uns in dieser Veranstaltung als Coaches, welche die Studierenden in der Entwicklung mit theoretischem Input und Diskussionen zu auftretenden Fragestellungen auf Augenhöhe unterstützen. Zusätzlich präsentierten wir Prozesse der Erkenntnisgewinnung für ausgewählte Aspekte im Unterricht. Wir planten explizit Raum für Diskussionen ein und standen in wöchentlich zwei Praktikumsstunden für Fragen zur Verfügung.

Dennoch war dieser Ansatz nicht ausreichend, denn die Arbeitsergebnisse legen nahe, dass die Studierenden aus der Diskussion keine Erkenntnisse gewinnen oder diese nicht transferieren konnten. Gründe hierfür könnten sein, dass die Studierenden nicht die Notwendigkeit von Diskussionen sehen und auch nicht die notwendigen Voraussetzungen mitbringen, um Diskussionen auf geeignetem Abstraktionsniveau zu führen oder sich daran zu beteiligen. Dadurch können sie dann auch keinen Erkenntnisgewinn daraus erlangen. Manche Studierende beurteilen in der Evaluierung vorgelebte Rückfragen im Diskurs als Inkompetenz:

*Es kommt sehr schlecht rüber, wenn es so ausschaut wie wenn Dozent X Hilfestunde für Dozent Y hält.*

Für uns bleibt hier die Frage, wie wir mit dieser Erkenntnis in Zukunft umgehen. Wie schaffen wir die nötigen Voraussetzungen zum Führen von Diskursen?

## 5.2 Umgang mit quantitativ vagen Aufgabenstellungen

Die Aufgabenstellungen waren insofern quantitativ vage formuliert als wir Dozierende für viele Fragen selbst auf Hypothesenbildung und nachträgliche Überprüfung in einer kritischen Diskussion angewiesen waren.

Beispiel: „Beschreiben Sie externe Schnittstellen soweit möglich.“ (als Aufgabenteilteil im Rahmen der Architekturdokumentation mit arc42). Wir konnten hier selbst keine genaue Aussage hinsichtlich der Anzahl an zu identifizierenden Schnittstellen machen. Allein das Fehlen einer quantitativen Aussage, sorgt unter den Studierenden für ein Gefühl von Unsicherheit. Daher forderten die Studierenden immer wieder quantitative Aussagen, wie viel sie zum Bestehen des Moduls machen müssen.

Eine Bewertung muss am Ende auch die Vorgehensweise berücksichtigen. Diese haben wir uns immer wieder beschreiben lassen. Wir haben uns an dem gängigen Bewertungssystem orientiert, nachdem eine Leistung eine Note 2 verdient, wenn sie voll den Anforderungen entspricht – also gemessen an dem, was in der Aufgabe von den Dozierenden gefordert war. Bei einer 1 muss die Leistung den Anforderungen in besonderem Maße und bei einer 3 im allgemeinen entsprechen. Wir mussten dabei keiner der Gruppen Mängel (Note 4) bescheinigen.

Im Laufe des Semesters stellten wir uns immer wieder die Frage: ob wir selbst das Projekt besser kennen müssten, um solche quantitativen Aussagen zu treffen. Wir kamen wiederholt zu dem Schluss, dass genau der fehlende Wissensvorsprung die Realitätsnähe in der Lehrveranstaltung herstellt. In der Evaluation am Semesterende zeigte sich ein sehr heterogenes Bild unter den Studierenden in Bezug auf ihre Fähigkeit zum Umgang mit den auftretenden Unsicherheiten:

*„Die Lehrveranstaltung ist sehr realitätsnah, wodurch meiner Meinung nach, die Relevanz der Inhalte gezeigt wird. Ich persönlich finde genau deswegen die VL sehr interessant.“*

*„Die Aufgaben waren nicht gut gestellt und es gab viele Diskussionen, was genau gemacht werden musste.“*

Wie schaffen wir trotz naturgemäß vager Arbeitsaufträge genügend Sicherheit für die Studierenden, sodass sie nicht zusätzlich in einer dauernden Unsicherheit hinsichtlich ihrer zu erwartenden Noten sind?

### 5.3 Einfluss der Unsicherheit auf die Qualität

Die Arbeitsergebnisse des Semesters zeigen, dass mit zunehmender Unsicherheit auch die Qualität in einigen Gruppen sinkt. Eine Gegenüberstellung der Aufgabenstellungen, deren Unsicherheit und die Qualität der Arbeitsergebnisse bekräftigt diese Beobachtung:

**Aufgabe *Glossar*: hoher Grad an Unsicherheit, schlechte Gesamtbewertung.** Zu Beginn des Semesters bot der Auftrag, ein Glossar anzulegen noch eine große Sicherheit für die Studierenden: Einige Begriffe aus dem Kontext des Projekts wurden in der Lehrveranstaltung ausführlich besprochen. Aber die Anzahl der expliziten Hinweise, Begriffe in das Glossar aufzunehmen, verringerte sich zügig. Studierende unterschätzten die Bedeutung von Begriffen aus dem fachlichen Kontext der Anwendung und erweiterten das Glossar kaum mehr. Die Begriffe waren natürlich auch in späteren Teilen der Lehrveranstaltung von Bedeutung und somit war die Orientierung in der Fachlichkeit mit zunehmender Unsicherheit behaftet, allein durch die Vernachlässigung des Aufschreibens. Alles in allem waren die entstandenen Glossare von minderer Qualität. Die Studierenden differenzierten unzureichend, welche Begriffe wichtig sind, viele Ergebnisse waren lediglich Abkürzungsverzeichnisse.

**API Dokumentation: niedriger Grad an Unsicherheit, gute Gesamtbewertung.** Die automatische Erstellung der API-Dokumentation war für die Studierenden mit wenig Unsicherheit verbunden. Sie wussten, welche Quelltextdateien sie analysieren sollten und welche Informationen zu extrahieren waren. Zusätzlich machten wir einen Vorschlag zur Darstellung der Informationen. Wir überließen es der Kreativität der Studieneden, in welcher Sprache sie den Vorgang automatisieren und wie sie die gewonnenen Informationen aufbereiten. Die Studierenden mussten also nicht unterscheiden, welche Informationen wichtig und welche unwichtig sind, die Aufgabe bot ein hohes Maß an Sicherheit. Die Ergebnisse der Studierenden waren durchwegs sehr gut zu bewerten.

**Schnittstellen: mittlerer Grad an Unsicherheit, mittlere Gesamtbewertung.** Eine Teilaufgabe der arc42-Dokumentation war die Identifikation und Beschreibung der externen Schnittstellen. Hier machten wir keine quantitative Aussagen, wie viele

Schnittstellen vorhanden sind und identifiziert werden können. Die Studienenden hatten also eine gewisses Maß an Unsicherheit, da sie selbst entscheiden mussten, wann sie ihre Recherchen beenden. Die Gesamtbewertung der Teilaufgabe viel im mittleren Bereich aus. Wir vermuten, dass die positive Tendenz sich durch die guten Suchmechanismen und die Verwendung von standardisierten Schnittstellen (z.B. http-Requests) erklären lässt.

**Datenbank: mittlerer Grad an Unsicherheit, mittlere Gesamtbewertung.** Bei der Beschreibung der Datenbankstruktur ist die Teilaufgabe zur Extraktion der Tabellen und Felder mit einer geringen Grad an Unsicherheit zu bewerten. Diese Informationen können mit einem üblichen Werkzeug automatisiert gewonnen werden. Alle Gruppen konnten diese Aufgabe mit einem guten Ergebnis absolvieren. Aufgrund mangelhaften Datenbankdesigns mussten die impliziten Primärschlüssel-Beziehungen aus dem Code retrospektiv herausgearbeitet werden. Dies bedeutete für die Studierenden ein hohes Maß an Unsicherheit, weshalb die Aufgabe im ganzen auch einen mittleren Grad an Unsicherheit aufweist. Nur wenige Gruppen fanden nachvollziehbare und stimmige Beziehungen. Wohl deshalb lagen die Leistungen dieser Aufgabe im mittleren Bereich.

**Laufzeitsicht: hoher Grad an Unsicherheit, schlechte Gesamtbewertung.** Hier steigt die Unsicherheit auf einen hohen Grad an, da die Studierenden die wichtigen von den unwichtigen Teilen der Funktion unterscheiden müssen und auch Erklärungen für zahlreiche *magic numbers* finden sollten. Diese Abstraktion und der Umgang der damit verbundenen Unsicherheit fiel den Studierenden sichtlich schwer. Großteils ähnelten die Aktivitätsdiagramme eins-zu-eins den Codezeilen im Sinne einer Nacherzählung. Nur einige wenige Gruppen beschrieben die Essenz der Funktion und fanden Erklärungen für die *magic numbers*. Damit ist die Gesamtbewertung der Ergebnisse als schlecht einzuordnen.

**Anforderungen: hoher Grad an Unsicherheit, schlechte Gesamtbewertung.** Ähnlich wie bei den Aktivitätsdiagrammen hatten die Studierenden große Probleme bei der Beschreibung der Anforderungen an ihre zuvor analysierten Funktionen. Zu 'raten' welche Anforderungen Grundlage für die Funktionen sind und diese im Kontext zu beschreiben viel allen Gruppen sichtlich schwer. Die Anzahl der Rückfragen waren während der Bearbeitungszeit auch auf dem Höhepunkt. Trotz zahlreicher Diskussionen und Unterstützung durch die Dozierenden konnte die Aufgabe nur mit einem eher schlechten Gesamtergebnis bewertet werden.

**UI Design: niedriger Grad an Unsicherheit, gute Gesamtbewertung.** Diese Aufgabe weißt einen niedrigen Grad an Unsicherheit auf, da der Funktionsumfang erhalten bleiben soll und Hinweise gegeben wurden. Alles in alle modernisierten alle Gruppen das Design und machten die Nutzung der Funktionen komfortabel und benutzerfreundlich. Das Gesamtergebnis kann damit als sehr gut bis gut bewertet werden.

Die Evaluationsergebnisse weisen auf die vorhandene Verunsicherung von Studierenden hin:

*„Projekt aus der Realität, auch wenn es nicht schön ist mit CF zu arbeiten“*

*„[...] das Projekt mit Lufthansa hat sich irgendwie nur bedingt geeignet, da man viele Herausforderungen/Aufgabenstellungen nur mir Raten lösen konnte“*

*„Der unübersichtliche Code und schlechter Codestil machen das analysieren des Projekts umständlich. Das mag eine akkurate Repräsentation der Wirklichkeit sein, motiviert aber nicht zum Ausarbeiten der Praktikumsaufgaben über das Minimum hinaus.“*

Unter den Studierenden, auch der höheren Semester, zeigt sich ein sehr heterogenes Spektrum. Die einen können mit der Unsicherheit der Aufgabenstellung umgehen – die anderen nicht. Folglich müssen wir als Dozierenden den Umgang mit Unsicherheit gezielt trainieren. Pair-Teaching ist aus unserer Sicht ein geeignetes Mittel um Diskurs vorzuleben und zu transportieren [ZBBK18, BUM09]. Doch es bleibt die Frage, wie die Kompetenz noch zusätzlich unterrichtet werden kann, wo es anscheinend nicht ausreicht, Raum für Diskussionen zu bieten und diese zu veranschaulichen.

Trainiert man den Umgang mit Unsicherheiten und den Diskurs gezielt in der Lehre, sollten diese Kompetenzen nach dem Prinzip des Constructive Alignment [Bi96] auch in adäquater Form geprüft werden. Doch wie bewertet man den Umgang mit Unsicherheiten oder das führen von Diskursen? Welche Kriterien gibt es, um die Kompetenz objektiv zu messen und die Bewertung transparent zu kommunizieren?

## 6 Diskussion und Ausblick

Wir haben in diesem Beitrag unsere Beobachtungen bei der Adressierung von Beurteilungsfähigkeit und dem Umgang mit Unsicherheiten am Beispiel der Veranstaltung Software-Archäologie vorgestellt.

Das Führen von Diskursen im (virtuellen) Hörsaal war unser Mittel, die Punkte in die Lehre zu integrieren. Diskurs braucht die Möglichkeit, unterschiedliche Standpunkte einnehmen zu können, von denen viele eine Berechtigung haben. Eine wichtige Erfahrung ist, dass ein gewachsenes umfangreiches und ohne klare Qualitätsstandards entwickeltes Fremdprojekt, gute Voraussetzungen dazu bietet. Die Firmenkooperation hat sich dafür als hilfreich erwiesen: das Industrie-Projekt kommt nicht aus des Professors „krauser Gedankenwelt“, wodurch gegenüber den Studierenden eine gefühlte Legitimierung, Glaubhaftigkeit und Authentizität hergestellt wird.

Trotzdem ist es schwierig, die Studierenden für einen Diskurs zu gewinnen. Unsere Studierenden drängen eher auf eindeutige Antworten oder Prozessbeschreibungen, die sie für die Prüfung verinnerlichen oder in der Studienarbeit anwenden können. Bieten

wir ihnen das nicht, fallen sie sie in eine große Unsicherheit. Hier entsteht ein Dilemma für die Dozierenden: durch zu viel und zu früh gegebenes Feedback passen sich die entstehenden Arbeitsergebnisse der Gedankenwelt der Dozierenden an – ein Resultat, das gerade vermieden werden soll. Das aus Studierendensicht „verweigerte“ Feedback nimmt ihnen gefühlt die Möglichkeit, ihre Noten im Verlauf des Semesters verbessern zu können.

Ermutigung, eingeschlagene Wege weiter zu verfolgen, Wertschätzung für Arbeitsergebnisse und die Diskussion der Vor- und Nachteile von Vorgehensweisen und Artefakten hilft, mit Unsicherheit im Projekt umzugehen. Sicherheit künstlich herzustellen fördert demgegenüber nicht die Fähigkeit, Unsicherheiten zu auszuhalten und damit auch im zukünftigen Berufsleben umgehen zu können.

Die Unterstützung der Studierenden in einem solchen Kurs, in dem höhere kognitive Fähigkeiten adressiert werden, übersteigt das Zeit-Budget, das für eine Lehrveranstaltung vorgesehen ist, erheblich. Hier fehlen effiziente Unterstützungsformen oder die Möglichkeit einer flexiblen Verrechnung von Arbeitsleistung hinsichtlich der Erfüllung der Lehrverpflichtung.

Zusammenfassend ergeben sich aus diesem Beitrag mehrere Fragen: Wie können wir Diskurs fördern, wie schaffen wir die nötigen Voraussetzungen zum Führen von Diskursen zu Software im Hörsaal? Welche weiteren Methoden eignen sich, um den Umgang mit Unsicherheit in die Lehre zu integrieren? Und: wie können wir diese Fähigkeiten objektiv messen, sodass die Messkriterien kommuniziert werden können? Dadurch könnte ein Faktor der Unsicherheit, nämlich die Unsicherheit bzgl. der Prüfungsleistungen verringert werden. Die Studierenden könnten sich dann auf die Unsicherheit fokussieren, die der Projektkontext mit sich bringt.

## Dank

Die Autorin/der Autor danken der Lufthansa Aviation Training für das zur Verfügung gestellte Projekt einschließlich aller Quelltexte.

## Literaturverzeichnis

[An01] Anderson, Lorin W.; Krathwohl, David R.; Airasian, Peter W.; Cruikshank, Kathleen A.; Mayer, Richard E.; Pintrich, Paul R.; Raths, James; Wittrock, Merlin C., Hrsg. A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives. Longman, New York, 1. Auflage, 2001.

[Bi96] Biggs, John: Enhancing teaching through constructive alignment. Higher education, 32(3):347–364, 1996.

[Bo09] Bower, Matt: Discourse analysis of teaching computing online. Computer Science Education, 19(2):69–92, 2009.

[BUM09] Böttcher, Axel; Utesch, Matthias; Moore, Austin: Erfahrungen mit Pair-Teaching für Software Engineering: Kooperation von Hochschule und Industrie. In: Tagungsband Software Engineering im Unterricht der Hochschulen (SEUH), Februar 2009 in Hannover. S. 5–15, 2009.

[De03] van Deursen, A.; Favre, J.-M.; Koschke, R.; Rilling, J.: Experiences in teaching software evolution and program comprehension. In: 11th IEEE International Workshop on Program Comprehension, 2003. S. 283–284, 2003.

[Du12] Durdik, Zoya; Klatt, Benjamin; Koziolek, Heiko; Krogmann, Klaus; Stammel, Johannes; Weiss, Roland: Sustainability guidelines for long-living software systems. In: 2012 28th IEEE International Conference on Software Maintenance (ICSM). S. 517–526, 2012.

[HP18] Hartung, M. J.; Pausch, R.: Soll man Studenten zwingen, im Hörsaal zu sitzen? Die Zeit, 73(2):61, 2018.

[HT02] Hunt, Andy; Thomas, Dave: Software archaeology. IEEE Software, 19(2):20–22, 2002.

[Le11] Lehner, Martin: Viel Stoff - wenig Zeit: Wege aus der Vollständigkeitsfalle. Haupt, Bern; Stuttgart; Wien, 2011.

[Na10] Nadel, Ben: Adobe ColdFusion Anthology: Clear and Concise Concepts from the Fusion Authority. Apress, Berkeley, CA, 2010.

[Pi17] Pinto, Gustavo Henrique Lima; Filho, Fernando Figueira; Steinmacher, Igor; Gerosa, Marco Aurelio: Training Software Engineers Using Open-Source Software: The Professors' Perspective. In: 2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T). S. 117–121, 2017.

[SH16] Starke, Gernot; Hruschka, Peter: arc42 in Aktion: Praktische Tipps zur Architekturdokumentation. Hanser, München, 2016.

[Si00] Sinn, Ulrich: Einführung in die klassische Archäologie. C. H. Beck, 2000.

[Sm14] Smith, Thérèse Mary; McCartney, Robert; Gokhale, Swapna S.; Kaczmarczyk, Lisa C.: Selecting Open Source Software Projects to Teach Software Engineering. In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education. SIGCSE '14, Association for Computing Machinery, New York, NY, USA, S. 397–402, 2014.

[TBS16] Thurner, Veronika; Böttcher, Axel; Schlierkamp, Katrin: Aligning learning objectives and exams: Moving upwards on the expertise level stack. In: IEEE Global Engineering Education Conference (EDUCON). S. 455–462, 2016.

[Th15] Thurner, Veronika; Böttcher, Axel; Schlierkamp, Katrin; Zehetmeier, Daniela: Lernziele für die Kompetenzentwicklung auf höheren Taxonomiestufen. In: Tagungsband Software Engineering im Unterricht der Hochschulen (SEUH), Dresden. S. 9–20, 2015.

[ZBBK18] Zehetmeier, Daniela; Böttcher, Axel; Brüggemann-Klein, Anne: Designing Lectures as a Team and Teaching in Pairs. In: Proc. 4th International Conference on Higher Education Advances (HEAd'18), Valencia. S. 873—880, 2018.

[Zu16] Zukunft, Olaf: , Empfehlungen für Bachelor- und Masterprogramme im Studienfach Informatik an Hochschulen (Juli 2016), 2016.