Tagungsband 14. Workshop

Software Engineering im Unterricht der Hochschulen

26. - 27. Februar 2015
HTW Dresden

Herausgeber:
Axel Schmolitzky, HAW Hamburg
Anna Sabine Hauptmann, HTW Dresden
http://ceur-ws.org/Vol-1332

Vorwort

Dieser Tagungsband begleitet den 14. Workshop „Software Engineering im Unterricht der Hochschulen“ an der HTW Dresden (SEUH 2015).

Seit der ersten SEUH im Jahre 1992 haben sich einige Eigenschaften der Tagung herausgebildet, die als SEUH-typisch bezeichnet werden können und die Jochen Ludewig, einer der Gründungsväter der Tagungsreihe, im Vorwort des Tagungsbandes der SEUH 2011 sehr schön aufgelistet hat: Neben Konstanten wie einem stabilen Termin (immer Ende Februar in den ungeraden Jahren), Deutsch als Tagungssprache, einer niedrigen Teilnahmegebühr und dem informellen Vorabendtreffen ist vor allem hervorzuheben, dass dies eine Tagung *zum intensiven Austausch zwischen Interessierten von Universitäten UND Fachhochschulen* ist. Um dies weiter gewährleisten zu können, haben wir in diesem Jahr darauf verzichtet, die SEUH erneut gemeinsam mit der SE durchzuführen – denn die findet in diesem Jahr erst in der zweiten Märzhälfte statt, zu spät für die meisten Kolleginnen und Kollegen an den Fachhochschulen. Solche und ähnliche Entscheidungen trifft der dreiköpfige Lenkungskreis, der für die Kontinuität der Tagung sorgen soll.

Der Tradition folgend, schafft auch die diesjährige Tagung einen Rahmen für die Diskussion zu aktuellen Fragen der Softwaretechnik im Kontext der Hochschullehre. Diese Fragen führen zu altbekannten Themen, die wiederholt Programmpunkte seit 1992 sind, aber auch zu neuen Themen, die sich insbesondere aus Veränderungen der industriellen Softwareentwicklung der letzten Jahre ergeben. Dies spiegeln auch die beiden eingeladenen Vorträge wider: Michael Kölling widmet sich der Schnittstelle zum Programmieren - einem gleichermaßen traditionellen wie auch essenziellen Thema der Softwaretechnik. Daniel Görsch skizziert aus der Perspektive heutiger industrieller Softwareentwicklung spezielle Herausforderungen, die somit auch Herausforderungen für unsere Absolventen sind. Beiden Anliegen ist u.a. gemein, dass sie das Programmieren als zentrales Handwerkszeug der Softwareentwicklung ansehen – und damit weiter dazu beitragen, dass Themen rund um das Programmieren-Lehren immer „salonfähiger“ werden.

Unter einem eigenständigen Programmpunkt demonstrieren drei studentische Gruppen von drei verschiedenen Hochschulen erfolgreiche Softwareprojekte, die im Kontext des Studiums entstanden sind.

Das Programmkomitee wählte aus 29 Einreichungen 12 Beiträge aus, die sich überwiegend folgenden Themenschwerpunkten zuordnen lassen:

Veränderungen der Softwaretechnikausbildung ausgehend von aktuellen Erkenntnissen der Lern- und Hirnforschung

- Warum ist eine Lernorientierung sinnvoll? Wie kann sie gelingen?
- Welche Rolle spielen Lernaufgabentypen?
- Welches sind die Lernziele für die Kompetenzentwicklung auf höheren Taxonomiestufen?
- Liefern die Kommunikation und Koordination zwischen Online-Rollenspielern nutzbare Erkenntnisse für Softwareprojekte?
- Erfordert die Ausrichtung auf Kompetenzorientierung die Neugestaltung der Softwaretechniklehre?

Softwareprojekte als tragender Bestandteil der Softwaretechnik

- Wie können Studierende frühzeitig und auf spielerische Weise die Sinnfälligkeit eines Projektmanagements erleben?
- Erweisen sich reale Aufgabenstellungen aus der Industrie als praktikable studentische Projekte?
- Welche Vor- und Nachteile hat ein monoedukatives Studienangebot „Informatik und Wirtschaft“?
- Ist es möglich, bereits den Start ins Studium mit Projekten zu gestalten?

Solide Programmierfähigkeiten als Grundlage der Entwicklung großer Softwaresysteme

- Wie kann die Sensibilität der Studierenden für guten Code gestärkt werden?
- Was unterstützt die Einführung in die Programmierung, damit Einstiegshürden und Gefahr der Überschätzung kleiner, Ergebnisse von Programmieranstrengungen sichtbarer, Kreativität und Motivation gefördert werden?

Wir hoffen, dass die ausgewählten Beiträge auch in diesem Jahr die Grundlage für anregende Diskussionen bilden.

Eine Tagung wie die SEUH ist nicht denkbar ohne die Menschen, die die Wichtigkeit von Diskussion und Dialog zwischen Gleichgesinnten erkannt haben und dies durch Ihre Unterstützung zum Ausdruck bringen. Ihnen wollen wir an dieser Stelle danken.

Zu allererst bedanken wir uns bei den Fachkolleginnen und Fachkollegen, die Beiträge einreichten und damit eine Auswahl ermöglichten. Dem Programmkomitee danken wir für die Begutachtung der Beiträge und deren Einordnung in das Programm.

Bei den Sponsoren bedanken wir uns, da ohne ihre Unterstützung die Durchführung des Workshops kaum möglich wäre. In diesem Jahr sind dies die Hamburger Firma *Workplace Solutions GmbH* und die Dresdner Firmen *Carl Zeiss Innovationszentrum GmbH*, *TraceTronic GmbH* und *Saxonia Systems AG*.

Christoph Zirkelbach von der HTW Dresden danken wir für die professionelle Gestaltung der Internetseite, für die Unterstützung bei der lokalen Organisation und bei der Erstellung dieses Tagungsbandes.

Weiterhin bedanken wir uns bei den studentischen Hilfskräften für ihre Unterstützung vor Ort.

Nicht zuletzt gilt auch der Hochschulleitung der HTW Dresden unser Dank, die uns Räumlichkeiten und technische Einrichtungen auf großzügige Weise zur Verfügung stellten. Ebenso bedanken wir uns bei der Fakultät Informatik/Mathematik der HTW Dresden für die vielfältige Unterstützung.

Den Mitarbeitern des Publikationsservices *CEUR Workshop Proceedings* danken wir für die elegante und zeitgemäße Möglichkeit, diesen Tagungsband zu veröffentlichen.

Aus technischer Perspektive hat sich das Konferenzsystem *EasyChair* bewährt.

Wir freuen uns, mit diesem Tagungsband einen Einblick in die SEUH 2015 geben zu können, und wünschen anregendes und inspirierendes Lesen.

Axel Schmolitzky, Hamburg
Anna Sabine Hauptmann, Dresden

Programmkomitee der SEUH 2015

Axel Böttcher, HAW München
Bernd Brügge, TU München
Martin Glinz, Uni Zürich
Anna Sabine Hauptmann, HTW Dresden
Maritta Heisel, Uni Duisburg-Essen
Stephan Kleuker, HS Osnabrück
Dieter Landes, HS Coburg
Claus Lewerentz, BTU Cottbus-Senftenberg
Horst Lichter, RWTH Aachen
Axel Schmolitzky, HAW Hamburg, Vorsitz
Kurt Schneider, Leibniz Uni Hannover
Simone Strippgen, BEUTH HS Berlin
Karin Vosseberg, HS Bremerhaven

Mitglieder des Lenkungskreises

Axel Schmolitzky, HAW Hamburg
Kurt Schneider, Leibniz-Universität Hannover
Andreas Spillner, Hochschule Bremen

Inhaltsverzeichnis

Eingeladene Vorträge

Beiträge

Eingeladene Vorträge

Die Schnittstelle zum Programmieren: Gedanken zum Re-Design von Programmierumgebungen

Michael Kölling, University of Kent, Canterbury, UK

M.Kolling@kent.ac.uk

Zusammenfassung

Das Design moderner Benutzerschnittstellen hat unsere Interaktion mit Softwaresystemen in den letzten zehn Jahren grundlegend neu erfunden. Dies gilt sowohl für Applikationen auf mobilen Geräten, als auch für Desktop-Programme. Unsere Programmierumgebungen, die wir selbst zur Softwarenentwicklung nutzen, haben sich allerdings wenig verändert. Der Großteil der Programmrepräsentation und -manipulation basiert immer noch auf reinem Text – eine eindimensionale Folge von Zeichen, zweidimensional auf dem Bildschirm arrangiert – mit Designelementen, die in Technologien vergangener Jahrzehnte verwurzelt sind.

In diesem Vortrag werden wir über die Zukunft von Softwarewerkzeugen spekulieren und präsentieren, wie ein moderner Editor zur Programmeingabe und -manipulation aussehen könnte.

Herausforderungen industrieller Softwareentwicklung

Daniel Görsch, Carl Zeiss Innovationszentrum für Messtechnik GmbH, Dresden

goersch@zeiss-izm.de

Zusammenfassung

Aus wissenschaftlicher Sicht stellen viele industrielle Softwareprojekte inklusive ihrer Entwicklungsprozesse und -methoden im besten Fall erfolgreiche „Fehlschläge“ dar.

Die Beobachtung der Interaktion der unterschiedlichen, in die Softwareentwicklung involvierten Interessengruppen legt nahe, dass auf diese Art und Weise ein Prozess induziert wird, der Ähnlichkeiten mit der Evolution biologischer Systeme aufweist. Ausgehend von dieser Analogie versucht dieser Vortrag, Ursachen für das besagte erfolgreiche „Scheitern“ herauszuarbeiten.

Entscheidend sind die herrschenden Rahmenbedingungen, unter denen Softwareentwicklung in der Industrie stattfindet. Dabei werden organisatorische, finanzielle und marktspezifische Aspekte unterschieden. In der Beziehung von Ursache und Wirkung bringen diese Rahmenbedingungen Ausprägungen von Eigenschaften hervor, welche charakteristisch für die industrielle Softwareentwicklung sind. Der Vortrag versucht diese Eigenschaften abzuleiten und damit eine Begründung für die Ausgangsdiagnose zu liefern.

Auf dieser Basis werden im letzten Teil des Vortrages Fähigkeiten und Kenntnisse formuliert, welche essentiell für eine erfolgreiche Arbeit in der industriellen Softwareentwicklung sind. Neben den fachlichen Kompetenzen sind es insbesondere psychosoziale Kompetenzen - wie Eigeninitiative und -motivation, die Fähigkeit zum systematischen, fokussierten Arbeiten und nicht zuletzt das interdisziplinäre Arbeiten und Kommunizieren, welche die Kunst des Programmierens alltagstauglich werden lassen.

Beiträge

Lernziele für die Kompetenzentwicklung auf höheren Taxonomiestufen

Veronika Thurner, Axel Böttcher, Kathrin Schlierkamp, Daniela Zehetmeier
Fakultät für Informatik und Mathematik, Hochschule München
<vorname>.<nachname>@hm.edu

Zusammenfassung

Nach unserer eigenen Lehrerfahrung ebenso wie der von zahlreichen Kolleginnen und Kollegen fällt es vielen Studierenden in MINT-Fächern schwer, höhere Kompetenzebenen gemäß der überarbeiteten Lernzieltaxonomie von Bloom (Anderson u. a., 2001) zu erreichen. Als Grundlage für die Konzeption von Lehr-/Lernmethoden, die für die Entwicklung dieser höheren Kompetenzebenen förderlich sind, definieren wir detaillierte Lernziele für die entsprechenden fachlichen Kompetenzen. Diese ergänzen wir um Lernzieldefinitionen für diejenigen Schlüsselkompetenzen, die eine essenzielle Voraussetzung dafür sind, dass diese fachlichen Kompetenzen auf der gewünschten Stufe überhaupt entwickelt werden können.

Motivation

Heutzutage müssen gute Softwerker mit anspruchsvollen Jobs in der Lage sein, in einem komplexen Umfeld und unterschiedlichsten Kontexten sicher zu agieren. Dafür müssen Sie sowohl über eine Fülle von nichtfachlichen Kompetenzen verfügen als auch fachlich hochqualifiziert sein (Böttcher u. a., 2011).

Um diesen fachlichen Anforderungen gerecht zu werden, müssen die Studierenden hochwertige kognitive Fähigkeiten entwickeln, die weit darüber hinaus gehen, einfach nur irgendwelches Wissen auswendig zu lernen und in der Prüfung wiederzugeben (Schaper u. a., 2012) (Bulimie-Lernen). Diese höheren Kompetenzebenen werden jedoch nicht ganz von alleine erreicht, sondern erfordern geeignete Lehr-/Lernansätze, die die angestrebten Kompetenzebenen gezielt entwickeln.

Grundlage für die Konzeption solcher Lehrmethoden ist die detaillierte Definition von entsprechenden Lernzielen, die genau beschreiben, was denn eigentlich eine bestimmte Fähigkeit auf einer bestimmten Kompetenzebene ausmacht. Diese muss insbesondere die fachlichen Fähigkeiten definieren, die die Studierenden entwickeln sollen.

Beim Definieren der angestrebten fachlichen Kompetenzen und aus den Erfahrungen unseres Lehralltages wurde deutlich, dass zwischen den fachlichen und den nicht-fachlichen Kompetenzen gewisse Abhängigkeiten bestehen. Insbesondere beobachten wir, dass ausgewählte Schlüsselkompetenzen wie z. B. das abstrakte Denken oder die Fähigkeit zur Selbstreflexion mit einem gewissen Reifegrad in den Studierenden ausgebildet sein müssen, bevor sie hochwertige fachliche Kompetenzen überhaupt entwickeln können.

Nicht alle Studierenden bringen jedoch diese erforderlichen Schlüsselkompetenzen in ausreichendem Maße mit, sondern müssen diese erst im Laufe ihres Studiums entwickeln. Daher ist es notwendig und sinnvoll, für ausgewählte Schlüsselkompetenzen ebenfalls Lernziele auf den verschiedenen Kompetenzebenen zu definieren, um greifbar zu machen, welche Fähigkeiten hier konkret erwartet werden bzw. zu entwickeln sind.

Zielsetzung

Um unsere Studierenden angemessen auf ihr späteres Arbeitsleben vorzubereiten, streben wir an, ihre fachlichen Kompetenzen bis hin zu den hochwertigen kognitiven Ebenen zu entwickeln. Dazu definieren wir für die Pflichtveranstaltungen „Softwareentwicklung 1 und 2“ aus dem ersten und zweiten Fachsemester der Studiengänge Bachelor Informatik und Bachelor Wirtschaftsinformatik *fachliche Lernziele* auf unterschiedlich anspruchsvollen Kompetenzebenen.

Des Weiteren definieren wir *Lernziele für ausgewählte Schlüsselkompetenzen*, die für die Entwicklung der höherwertigen fachlichen Lernziele essenziell erforderlich sind. Diese Lernziele sind die Grundlage für die Konzeption von Lehrmethoden und Interventionen, die den Auf- und Ausbau dieser zentralen Schlüsselkompetenzen in den Studierenden gezielt fördern.

State of the Art

Orientiert an (Schott u. Ghanbari, 2009) verstehen wir unter *Kompetenzen* diejenigen Eigenschaften und Fähigkeiten, die erforderlich sind, um eine bestimmte Menge und Art von Aufgaben sinnvoll ausführen zu können. Dabei kategorisieren wir Kompetenzen in

Anlehnung an (Chur, 2004) und (Schaeper u. Briedis, 2004) in die vier Bereiche *Selbst-*, *Methoden-*, *Sozial-* und *Fachkompetenz*.

Als *Schlüsselkompetenzen* bezeichnen (In der Smitten u. Jaeger, 2009) diejenigen Kompetenzen, die die spezifisch-fachlichen Fähigkeiten so ergänzen, dass eine Person damit ihren eigenen Bedürfnissen gerecht werden, in Gemeinschaft mit anderen leben und einer nützlichen und einkommenssichernden Arbeit nachgehen kann.

Ein bekannter Ansatz zur Beschreibung von Lernzielen im kognitiven Bereich ist die Lernzieltaxonomie von Bloom (Bloom u. a., 1956), die Kompetenzen auf verschiedenen Fähigkeitsebenen definiert. Dieser Ansatz wurde später insbesondere hinsichtlich der höheren Kompetenzebenen von (Anderson u. a., 2001) überarbeitet.

Im Folgenden orientieren wir uns an dieser überarbeiteten Version, mit den Kompetenzstufen *1: Erinnern*, *2: Verstehen*, *3: Anwenden*, *4: Analysieren*, *5: Evaluieren* und *6: Kreieren*.

Lernziele für Fachkompetenzen

Die nachfolgenden Tabellen definieren Lernziele für die fachlichen Inhalte, die im Fokus der Lehrveranstaltungen „Softwareentwicklung 1“ bzw. „Softwareentwicklung 2“ stehen, die an der Fakultät für Informatik und Mathematik der Hochschule München durchgeführt werden. In den einzelnen Spalten werden bewusst nicht die Verben aus den Spaltenüberschriften (den Kompetenzebenen nach der überarbeiteten Lernzieltaxonomie von Bloom) verwendet. Stattdessen werden Verben gewählt, die ein von außen beobachtbares Ergebnis nach sich ziehen. Dadurch wird die Testbarkeit der Lernziele gewährleistet.

Da die Kompetenzen auf den verschiedenen Ebenen für viele der fachlichen Inhalte vom Prinzip her ähnlich gelagert sind, definieren Tabellen 1 und 2 (oben) die Lernziele generisch im Sinne einer Schablone, in deren Formulierung bei Bedarf konkrete programmiersprachliche Konstrukte bzw. Artefakte einzusetzen sind.

Ergänzend werden in Tabellen 1 und 2 (mitte) Lernziele für Qualitätskriterien für Software definiert, da diese Qualitätskriterien als Maßstab für die generischen Lernziele für Softwareentwicklung benötigt werden. Für den Bereich der Qualitätskriterien liegen die Lernziele der Level 5 (Evaluieren) und 6 (Kreieren) dabei nicht mehr im Fokus der Lehrveranstaltungen „Softwareentwicklung 1 und 2“.

Die Lernzieldefinitionen in Tabellen 1 und 2 (unten) sowie 3 und 4 (oben) zeigen exemplarisch für die fachlichen Inhalte *Kontrollstrukturen* und *Algorithmen*, wie die generische Definition der Lernziele für *Softwareentwicklung* schematisch auf programmiersprachliche Konstrukte übertragen wird. Nach dem gleichen Schema verlaufen die Lernzieldefinitionen für alle anderen programmiersprachlichen Konstrukte, die in der Lehrveranstaltung behandelt werden, insbesondere für die folgenden inhaltlichen Themenbereiche: Klassen, Datentypen, Variablen, primitive Datentypen, Sichtbarkeit, Pakete, Zeichenketten, Arrays, Rekursion, Listen, Vererbung, Exception Handling, Generische Datentypen und Collections. Da diese Lernzieldefinitionen hochgradig redundant mit der generischen Definition verlaufen, bieten sie keinen nennenswerten Mehrwert gegenüber der generischen Definition. Entsprechend wird hier auf eine weitere Ausarbeitung für andere programmiersprachliche Konstrukte verzichtet.

Einige der fachlichen Inhalte, die in der Lehrveranstaltung „Softwareentwicklung“ behandelt werden, sind jedoch prinzipiell anders gelagert als die programmiersprachlichen Konstrukte. Entsprechend erfordern deren zugehörige Kompetenzen daher eine komplett eigenständige Lernzieldefinition, die sich nicht schematisch aus der generischen Lernzieldefinition ableiten lässt.

Die Programmentwicklung ist eine ausgeprägt konstruktive Tätigkeit, bei der ein (hoffentlich) lauffähiges System erstellt und somit etwas geschaffen wird. Im Gegensatz dazu ist *Testen* vom Prinzip her eine destruktive Tätigkeit (auch wenn das Ziel des Testens ist, dass das geschaffene System am Ende stabil und damit qualitativ besser wird). Beim Testen geht es darum, wunde Punkte zu identifizieren und diese sichtbar zu machen. Dieses prinzipiell andere Ziel erfordert andere Ansätze und damit auch andere Kompetenzen als der zu entwickelnde Anteil der Softwareentwicklung (siehe Tabellen 3 und 4 (Mitte)). Die Lernziele auf Level 6 (Kreieren) liegen dabei nicht mehr im Fokus der Lehrveranstaltungen „Softwareentwicklung 1 und 2“.

Debugging wiederum, also die systematische Fehlersuche, ist eine methodenintensive Fachlkompetenz (siehe Tabellen 3 und 4 (unten)). Auch hier wird im Gegensatz zur eigentlichen Programmierung kein Produkt gebaut, das zwar virtuell, aber trotzdem irgendwie greifbar ist. Statt dessen geht es beim Debugging eher um die Anwendung von Werkzeugen und Strategien zum Finden und Beseitigen von Fehlern, die nach erfolgreichem Debugging hoffentlich nicht mehr da sind. Entsprechend sind auch hier potenziell messbare Ergebnisse der einzelnen Kompetenzstufen grundsätzlich anders geartet als bei den programmiersprachlichen Konstrukten, sodass eine spezifische Lernzieldefinition für den Themenbereich des Debuggings erforderlich ist. Auch in diesem fachlichen Bereich liegen die Lernziele auf Level 6 (Kreieren) nicht mehr im Fokus der Lehrveranstaltungen „Softwareentwicklung 1 und 2“.

Zusammenhang zwischen Fach- und Schlüsselkompetenzen

In der Lehrpraxis ist zu beobachten, dass viele Studierende sich nicht nur beim Erlernen spezifischer fachlicher Inhalte schwer tun, sondern dass das prin-

Tabelle 1: Definition generischer und konkreter fachlicher Lernziele für Softwareentwicklung (Teil 1)

Fach-kompetenz	Stufe 1: Erinnern Die Studierenden...	Stufe 2: Verstehen	Stufe 3: Anwenden
Generisch für Software-entwicklung	... definieren die Grundbegriffe des jeweiligen fachlichen Inhalts. ... benennen in einem vorgegebenen Artefakt (Anforderungsdefinition, Testfälle, Entwurf, Algorithmusspezifikation, Quelltext) die dort verwendeten Konstrukte/Elemente mit den korrekten Fachbegriffen. ... schreiben die konkrete Syntax eines programmiersprachlichen Konstruktes korrekt auf und halten dabei die Syntaxkonventionen ein.	... erklären in eigenen Worten die Bedeutung der Grundbegriffe des jeweiligen fachlichen Inhalts, insbesondere der programmiersprachlichen Konstrukte. ... beschreiben in eigenen Worten die Unterschiede zwischen den einzelnen programmiersprachlichen Konstrukten. ... begründen, welches programmiersprachliche Konstrukt in welchem Kontext zu verwenden ist, und warum. ... begründen, warum Softwareentwicklung aus mehr Schritten besteht als nur der Implementierung.	... setzen einen textuell oder grafisch vorgegebenen Entwurf in Quelltext einer festgelegten Programmiersprache um. Der Entwurf gibt dabei die Struktur der Klassen incl. von deren Attributen und Methoden vor. Für die Methoden ist der Algorithmus in seinen Grundzügen ebenfalls vorgegeben. Der Quelltext erfüllt dabei grundlegende Qualitätsanforderungen (Lesbarkeit, Testbarkeit, Korrektheit). ... ermitteln zu einer gegebenen Implementierung und konkreten Eingabe- bzw. Startwerten das konkrete Ergebnis.
Qualitäts-kriterien für Software	... benennen grundlegende Qualitätskriterien für Software, z.B. Lesbarkeit, Testbarkeit, Korrektheit und Effizienz. ... definieren die Bedeutung der Qualitätskriterien.	... erklären in eigenen Worten die Bedeutung der einzelnen Qualitätskriterien. ... beschreiben, warum diese Qualitätskriterien wichtig sind und welche Folgen es hat, wenn sie nicht eingehalten werden.	... halten bei der Softwareentwicklung die vorgegebenen Qualitätskriterien ein.
Kontroll-strukturen	... benennen verschiedene Arten von Kontrollstrukturen. ... definieren für jede Art von Kontrollstruktur deren Bedeutung. ... benennen in einem vorgegebenen Quelltext die dort verwendeten Kontrollstrukturen mit den korrekten Fachbegriffen. ... schreiben für jede Art von Kontrollstruktur deren konkrete Syntax auf und halten dabei die Syntaxkonventionen ein.	... erklären in eigenen Worten die Bedeutung der verschiedenen Kontrollstrukturen. ... beschreiben in eigenen Worten die Unterschiede zwischen for-, while- und do-while-Schleife. ... beschreiben in eigenen Worten die Unterschiede zwischen if-, if-else- und switch-Fallunterscheidungen. ... begründen, welche Kontrollstruktur in welchem Kontext zu verwenden ist, und warum.	... setzen eine textuell oder grafisch vorgegebene Algorithmusspezifikation in Quelltext einer festgelegten Programmiersprache um und verwenden dabei die passenden Kontrollstrukturen. Der Quelltext erfüllt dabei grundlegende Qualitätsanforderungen (Lesbarkeit, Testbarkeit, Korrektheit). ... werten zu einer gegebenen Implementierung die Kontrollstrukturen korrekt aus, indem sie zu konkreten Eingabe- bzw. Startwerten das konkrete Ergebnis ermitteln.

Tabelle 2: Definition generischer und konkreter fachlicher Lernziele für Softwareentwicklung (Teil 2)

Stufe 4: Analysieren **Die Studierenden...**	**Stufe 5: Evaluieren**	**Stufe 6: Kreieren**
... geben zu einer vorgegebenen Implementierung an, was diese prinzipiell macht, abstrahiert von konkreten Eingabe- bzw. Startwerten. ... ermitteln aus einer informell gegebenen Problemformulierung die Anforderungen und dokumentieren diese präzise in einer angemessenen Form (grafisch oder textuell). ... identifizieren in einer gegebenen Anforderungsbeschreibung Strukturen und Zusammenhänge und dokumentieren diese präzise in einer angemessenen Form (grafisch oder textuell).	... wägen systematisch ab, welches Konzept bzw. Konstrukt der Programmiersprache am besten geeignet ist, um eine bestimmte Anforderung umzusetzen. ... identifizieren Stärken und Verbesserungspotenzial in einem gegebenen Artefakt (Anforderungsspezifikation, Testfälle, Entwurf, Algorithmusspezifikation, Quelltext) oder in der Problemformulierung bzw. Anforderungsbeschreibung des Kunden. ... bewerten ihre eigene Lösung (d.h. ein von ihnen selbst erstelltes Artefakt) kritisch auf Stärken und Schwächen, die hinsichtlich grundlegender Qualitätsanforderungen bestehen (Lesbarkeit, Testbarkeit, Korrektheit).	... entwickeln für ein einfaches Problem aus einer gegebenen Anforderungsspezifikation heraus einen Entwurf, der sowohl die Gesamtstruktur der Lösung als auch die einzelnen Algorithmen vorgibt. Der Entwurf erfüllt dabei grundlegende Qualitätsanforderungen (Korrektheit, Effizienz der Algorithmen, Testbarkeit). (Ein „einfaches Problem“ ist dabei eine Aufgabenstellung, die mit maximal zehn Klassen objektorientiert zu lösen ist. Für komplexere Probleme ist diese Kompetenz Lernziel der Veranstaltung „Software Engineering“.)
... untersuchen, inwieweit eine bestehende Software die vorgegebenen Qualitätsanforderungen erfüllt. ... identifizieren Verstöße gegen die vorgegebenen Qualitätskriterien.	... bewerten kritisch, ob die bestehenden Qualitätskriterien für ein gegebenes Anwendungsszenario angemessen und ausreichend sind.	... definieren selbst neue, eigene Qualitätskriterien für neuartige Anwendungsszenarien.
... geben zu einer vorgegebenen Folge/Konstruktion aus Kontrollstrukturen an, was diese prinzipiell macht, abstrahiert von konkreten Eingabe- bzw. Startwerten. ... ermitteln aus einer gegebenen Anforderungsbeschreibung, mit welchen Kontrollstrukturen die Anforderung prinzipiell erfüllt werden kann.	... wägen systematisch ab, welche Kontrollstruktur am besten geeignet ist, um eine bestimmte Anforderung umzusetzen. ... identifizieren Stärken und Verbesserungspotenzial in einem gegebenen Artefakt (Algorithmusspezifikation, Quelltext). ... bewerten ihre eigene Lösung (d.h. ein von ihnen selbst erstelltes Artefakt) kritisch auf Stärken und Schwächen, die hinsichtlich grundlegender Qualitätsanforderungen bestehen (Lesbarkeit, Testbarkeit, Korrektheit).	... entwickeln für ein einfaches Problem aus einer gegebenen Anforderungsspezifikation heraus einen Algorithmus oder Quelltext, der dazu die geeigneten Kontrollstrukturen verwendet. Dieser erfüllt dabei grundlegende Qualitätsanforderungen (Korrektheit, Effizienz der Algorithmen, Testbarkeit).

Tabelle 3: Beispiele für konkrete fachliche Lernziele für Softwareentwicklung (Teil 1)

Fach-kompetenz	Stufe 1: Erinnern Die Studierenden...	Stufe 2: Verstehen	Stufe 3: Anwenden
Algorithmen	... definieren den Begriff Algorithmus. ... benennen mindestens eine formale, nicht programmiersprachliche Notation zum Formulieren von Algorithmen.	... erklären in eigenen Worten die Bedeutung des Begriffs Algorithmus. ... begründen, warum formale Notationen zur Beschreibung von Algorithmen notwendig sind.	... setzen eine textuell oder grafisch vorgegebene Algorithmusspezifikation in einer festgelegten Programmiersprache um. ... führen einen gegebenen Algorithmus für vorgegebene Startwerte aus und ermitteln das Ergebnis. (Bsp.: Für die Startwerte 18 und 24 liefert der Algorithmus den Wert 6 als Ergebnis.)
Unit-Testen	... definieren den Begriff Unit-Testen. ... benennen verschiedene assert-Methoden, die in JUnit-Tests verwendet werden können. ... benennen die verschiedenen Annotationen für Unit-Tests. ... schreiben die konkrete Syntax für Testklassen und Testmethoden auf und halten dabei die Syntaxkonventionen ein.	... begründen, warum automatisiertes Testen notwendig und sinnvoll ist. ... beschreiben in eigenen Worten die Bedeutung und den qualitätssichernden Beitrag einer vollständigen Testabdeckung.	... erstellen schematisch grundlegende Testfälle. ... setzen systematisch Werkzeuge ein, die den Grad der erreichten Testabdeckung ermitteln. ... nutzen ein Werkzeug wie z.B. JUnit, um Unit-Tests automatisiert auszuführen.
Debugging	... definieren den Begriff Debugging. ... benennen verschiedene Ansätze für das Debugging. ... benennen verschiedene Grundfunktionen eines Debugging-Werkzeuges.	... erklären in eigenen Worten, warum die Verwendung eines Debugging-Werkzeuges sinnvoll ist	... verwenden den Debugger schematisch, um das Programmverhalten zu visualisieren. ... gleichen beim Verwenden des Debuggers das, was der Debugger anzeigt, ab mit der eigenen mentalen Erwartung, bis beides nicht mehr zueinander passt und zeigen so Soll-/Ist-Differenzen auf.

Tabelle 4: Beispiele für konkrete fachliche Lernziele für Softwareentwicklung (Teil 2)

Stufe 4: Analysieren **Die Studierenden...**	**Stufe 5: Evaluieren**	**Stufe 6: Kreieren**
... geben zu einem vorgegebenen Algorithmus an, was dieser prinzipiell macht, abstrahiert von konkreten Eingabe- bzw. Startwerten. (Bsp.: Der Algorithmus ermittelt den größten gemeinsamen Teiler zweier natürlichen Zahlen.)	... wägen systematisch ab, welches Konzept bzw. Konstrukt der Programmiersprache am besten geeignet ist, um eine bestimmte Anforderung in einem Algorithmus umzusetzen. ... identifizieren Stärken und Verbesserungspotenzial in einer gegebenen Algorithmusspezifikation. ... bewerten einen von ihnen selbst erstellen Algorithmus kritisch auf Stärken und Schwächen, die hinsichtlich grundlegender Qualitätsanforderungen bestehen (Lesbarkeit, Testbarkeit, Korrektheit).	... entwickeln für ein einfaches Problem aus einer gegebenen Anforderungsspezifikation heraus einen Algorithmus. Dieser erfüllt dabei grundlegende Qualitätsanforderungen (Korrektheit, Effizienz, Testbarkeit).
... untersuchen, welche Rand- und Normalfälle in einer gegebenen Aufgabenstellung auftreten, fassen diese zu geeigneten Äquivalenzklassen zusammen und definieren dazu passende Testfälle als Repräsentanten.	... bewerten, ob die Menge und Art der formulierten Tests zur Qualitätssicherung ausreichend ist. (Die werkzeuggemessene Testabdeckung ist nur ein Teil dieses Bewertungsprozesses.)	... entwickeln eigene Teststrategien für komplexe, neuartige Problemstellungen. ... entwickeln eine neue Testinfrastruktur.
... ziehen Rückschlüsse aus den im Debugger erkannten Differenzen aus Soll und Ist. Dabei suchen sie nach der Ursache der beobachteten Differenzen und ermitteln die Stelle, an der die Ursache vermutet wird und an der somit die Lösung ansetzen muss.	... bewerten, ob eine gegebene, selbst angewendete oder beobachtete Vorgehensweise zur Fehlersuche zielführend und effizient ist.	... entwickeln neuartige Strategien zur Suche und Eingrenzung von Softwarefehlern.

zipielle Erreichen einer bestimmten Kompetenzebene Probleme aufwirft, unabhängig vom konkreten fachlichen Inhalt. Dies führte zu der These, dass nicht ausschließlich fachliche Defizite dafür verantwortlich sind, dass Studierende eine gewünschte höhere Kompetenzebene für einen konkreten fachlichen Inhalt gar nicht oder nur eingeschränkt entwickeln können. Vielmehr müssen also bestimmte Schlüsselkompetenzen in den Studierenden bereits bis zu einem gewissen Grad entwickelt sein, damit insbesondere die höheren Ebenen der überarbeiteten Lernzieltaxonomie von Bloom überhaupt erreichbar sind.

Um diese These zu belegen haben wir eine Vielzahl von Fehlern analysiert, die beim Erlernen des Programmierens mit großer Häufigkeit auftreten, basierend sowohl auf Fehlern aus der Literatur (wie z. B. (Hristova u. a., 2003; Humbert, 2006; Kaczmarczyk u. a., 2010; Sirkiä u. Sorva, 2012; Sorva, 2008)) als auch auf eigenen Beobachtungen. Diese Fehler haben wir anschließend kategorisiert und den entsprechenden Kompetenzebenen nach der überarbeiteten Lernzieltaxonomie von Bloom zugeordnet. Darauf aufbauend wurden die Ursachen dieser Fehler analysiert und kategorisiert. Dabei wurde deutlich, dass bestimmte Kompetenzebenen bestimmte Grundfähigkeiten essenziell erfordern (Zehetmeier u. a., 2015).

Beispielsweise setzt die Ebene 4 *Analysieren* zwingend die Fähigkeit des analytischen Denkens voraus, sowie (zumindest im informatischen Kontext) abstraktes Denken. Ebene 5 *Evaluieren* erfordert unter anderem kritisches Hinterfragen sowie ggf. die Fähigkeit zur Selbstreflexion, und Ebene 6 *Kreieren* benötigt ein gewisses Maß an Kreativität. Tabelle 5 fasst die Ergebnisse zusammen, in Erweiterung von (Zehetmeier u. a., 2015).

Wenn wir mit unseren Studierenden insbesondere die höheren Kompetenzebenen nach der überarbeiteten Lernzieltaxonomie von Bloom erreichen wollen, müssen diese grundlegenden Schlüsselkompetenzen also zunächst in ausreichendem Umfang vorhanden sein, bzw. bei Bedarf entsprechend entwickelt werden. Erst wenn hier die notwendigen Voraussetzungen geschaffen sind macht es Sinn, sich auf die entsprechenden fachlichen Inhalte zu fokussieren, weil diese sonst ohnehin nicht effektiv gelernt werden können.

Die Frage, inwieweit die Entwicklung der erforderlichen Schlüsselkompetenzen im Aufgabenbereich der Hochschullehre liegt, ist nicht einfach zu beantworten. Eine umfassende Diskussion der möglichen Sichtweisen liegt nicht im Fokus dieser Arbeit.

Lernziele für Schlüsselkompetenzen

Unabhängig vom konkreten fachlichen Inhalt sind also bestimmte Schlüsselkompetenzen erforderlich, um fachliche Kompetenzen auf einer gewünschten höheren Ebene gemäß der überarbeiteten Lernzieltaxonomie von Bloom überhaupt entwickeln zu können.

Um greifbar zu machen, was diese Schlüsselkompetenzen im informatik-nahen Lernkontext genau bedeuten und welches Maß an Fähigkeiten als erforderlich vorausgesetzt bzw. bei Bedarf zu entwickeln ist, definieren wir im Folgenden auch für einige Schlüsselkompetenzen Lernziele gemäß der überarbeiteten Lernzieltaxonomie von Bloom. Dabei fokussieren wir lediglich eine Auswahl an relevanten Schlüsselkompetenzen, mit jeweils einem Repräsentanten aus den Bereichen der Selbst-, Sozial- und Methodenkompetenzen.

Auch hier liegen jeweils die Lernziele auf Level 6 (Kreieren) der einzelnen ausgewählten Schlüsselkompetenzen nicht mehr im Fokus der Lehrveranstaltungen „Softwareentwicklung 1 und 2“.

Selbstkompetenz: Selbstreflexion

„Einsicht ist der erste Schritt zur Besserung“, sagt ein deutsches Sprichwort. Dahinter steckt die Idee, dass ich ein Problem selbst nur dann beheben kann, wenn ich es erkannt habe und darüber nachdenken kann.

In diesem Sinne ist die Fähigkeit zur *Selbstreflexion* also zentraler Bestandteil eines effizienten Lernprozesses und damit eine Selbstkompetenz, die für den Studienerfolg, das spätere Arbeitsleben sowie allgemein auch für die persönliche Weiterentwicklung im privaten Leben von zentraler Bedeutung ist. Gleichzeitig zeigt unsere Lehrerfahrung aus den letzten Jahren, dass die Fähigkeit zur Selbstreflexion bei unseren Studierenden zu Beginn des Studiums wenig bis gar nicht ausgeprägt ist. Daher fokussieren wir diese Kompetenz in besonderem Maße als eine Schlüsselkompetenz im Sinne der Studierfähigkeit.

Tabellen 6 und 7 (oben) definieren für die Fähigkeit der Selbstreflexion Lernziele auf den verschiedenen Kompetenzebenen. Wünschenswert im Sinne der Studierfähigkeit wäre eine Kompetenzebene von mindestens Stufe 4 (Analysieren). Viele unserer Studierenden (insbesondere diejenigen, die sich fachlich schwer tun) verfügen jedoch lediglich über Stufe 1 (Erinnern) dieser Fähigkeit. D. h. ihnen ist zwar immerhin der Begriff vage bekannt. Ein Grundverständnis dafür, inwieweit Selbstreflexion den eigenen Lernprozess unterstützt, ist dagegen bei diesen Personen meist nicht vorhanden.

Sozialkompetenz: Kritikfähigkeit

Gerade dann, wenn die Fähigkeit zur Selbstreflexion erst unzureichend entwickelt ist, ist Feedback von außen oft ein notwendiges Mittel, um Stärken und Schwächen bewusst zu machen und so eine Weiterentwicklung effektiv auf den Weg zu bringen. Diese „Weiterentwicklung von außen“ setzt jedoch voraus, dass der Empfänger mit dem erhaltenen Feedback angemessen umzugehen weiß. Entsprechend ist *Kritikfähigkeit* in der Rolle der Kritikempfangenden wichtiger Bestandteil der Studierfähigkeit.

In vielen Veranstaltungen arbeiten die Studierenden in Teams zusammen. Darüber hinaus streben viele

Tabelle 5: Schlüsselkompetenzen, die zum Erreichen einer fachlichen Kompetenzebene notwendig sind (Auswahl)

Stufe	Kompetenz-ebene	Fehlerklasse	Erforderliche Schlüsselkompetenzen
6	**Kreieren**	**Mangel an Innovation**	Kreativität Abstraktes Denken (Neuartige Gesetzmäßigkeiten finden)
5	**Evaluieren**	**Qualitätslücke**	Analytisches Denken (Kritische Punkte identifizieren) Abstraktes Denken (Relevanz der kritischen Punkte werten) Kritisches Hinterfragen Selbstreflexion (Bei Evaluation eigener Ergebnisse) Pragmatismus (Kosten-Nutzen-Entscheidung fällen)
4	**Analysieren**	**Unstrukturiertheit**	Analytisches Denken (Bestandteile identifizieren, Zusammenhänge erkennen) Abstraktes Denken (Sinn der Zusammenhänge werten)
3	**Anwenden**	**Falsche Entscheidungen**	Abstraktes Denken (Passende Abstraktionsstrategie wählen) Analytisches Denken (Problem klassifizieren) Entscheidungsfähigkeit (Lösungsansatz wählen)
2	**Verstehen**	**Fehlvorstellung**	Abstraktes Denken (Regelwerk erkennen und anwenden) Analytisches Denken (Zusammenhänge nachvollziehen) Ganzheitliches Denken (Mit Vorwissen verbinden)
1	**Erinnern**	**Wissenslücke**	Durchhaltevermögen Fleiß Selbstreflexion (Eigenen Lerntyp kennen, eigene Fähigkeiten einschätzen)
0	**–**	**Geistiger Tippfehler** „Mental Typo“ nach D. E. Knuth (Knuth, 1989)	Sorgfalt Konzentration Belastbarkeit

Tabelle 6: Definition von Lernzielen für Schlüsselkompetenzen (Teil 1)

Schlüssel-kompetenz	Stufe 1: Erinnern Die Studierenden...	Stufe 2: Verstehen	Stufe 3: Anwenden
Selbst-reflexion	... definieren den Begriff Selbstreflexionsfähigkeit.	... erklären in eigenen Worten die Bedeutung von Selbstreflexion im Lernprozess und für die persönliche Weiterentwicklung. ... erklären in eigenen Worten die Bedeutung von Reflexion für die Fähigkeit des kritischen Hinterfragens und die Fähigkeit des Evaluierens allgemein.	... setzen sich anhand eines gegebenen Reflexionsleitfadens mit einer eigenen Situation, Vorgehensweise oder einem selbst erstellten Artefakt auseinander und dokumentieren ihre Erkenntnisse auf der Ebene von beobachtbaren Symptomen (nicht von Ursachen).
Kritik-fähigkeit	... definieren den Begriff Kritikfähigkeit. ... benennen die beiden Rollen, die eine Person im Kontext von Kritik einnehmen kann. ... benennen die zentralen Regeln für das Geben und Empfangen von Feedback.	... begründen die Stringenz der Feedback-Regeln für Feedback-Geber und Feedback-Nehmer. ... beschreiben in eigenen Worten die Bedeutung von Kritik für die persönliche Weiterentwicklung.	... formulieren als Feedback-Geber ihre Kritik gemäß der Feedback-Regeln. (Das Identifizieren der Kritikpunkte, die hier kommuniziert werden, kommt dabei aus einem anderen Kompetenzbereich, beispielsweise aus der „Evaluieren“-Spalte einer fachlichen Kompetenz.) ... halten als Feedback-Nehmer beim Empfangen von Kritik die formalen Feedback-Regeln ein, hören also zu und rechtfertigen sich nicht.
Abstraktes Denken	... definieren die Begriffe Abstrahieren und Konkretisieren. ... definieren die Kompetenz des abstrakten Denkens. ... definieren den Begriff Abstraktionsebene und die Zusammenhänge zwischen konkreter Ebene, Typ-Ebene und Meta-Ebene.	... erklären, dass es in der Informatik überwiegend um abstrakte Inhalte geht und abstraktes Denken daher in diesem Bereich eine unerlässliche Fähigkeit ist. ... begründen, dass jegliches Verstehen eines Sachverhaltes voraussetzt, dass das grundlegende Regelwerk bzw. die grundlegenden Gesetzmäßigkeiten verstanden wurden. (Bsp: Auch wenn ich weiß, dass 2+3=5 gilt kann ich 3+4=? trotzdem nur lösen, wenn ich das grundlegende Regelwerk verstanden habe.)	... leiten aus einer gegebenen einfachen Abstraktion bzw. einem gegebenen einfachen Regelwerk konkrete Dinge und Aussagen ab. ... extrahieren aus einer gegebenen Menge von konkreten, einfachen Beispielen das zugrunde liegende, ebenfalls noch einfache Regelwerk. ... ermitteln aus einer gegebenen Menge von nichttrivialen Beispielen anhand eines vorgegebenen Leitfadens das zugrunde liegende Regelwerk. ... lesen aus einem vorgegebenen Metamodell Aussagen über die darunter liegende Typ-Ebene heraus.

Tabelle 7: Definition von Lernzielen für Schlüsselkompetenzen (Teil 2)

Stufe 4: Analysieren **Die Studierenden...**	**Stufe 5: Evaluieren**	**Stufe 6: Kreieren**
... bestimmen ein Ziel für die Selbstreflexion, also eine tiefer greifende Erkenntnis, die mittels Selbstreflexion gewonnen werden soll. ... erarbeiten die Ursachen der beobachteten Symptome. ... identifizieren Maßnahmen, die geeignet sind, um die Symptome zukünftig zu vermeiden (falls diese negativ waren) oder erneut zu provozieren (falls diese positiv waren); im Sinne von „Lessons Learned“.	... bewerten, ob ein gegebenes Konzept der Selbstreflexion für das von ihnen angestrebte Ziel der Reflexionsarbeit hilfreich ist. ... beobachten ihren eigenen Reflexionsprozess und bewerten, inwieweit dieser stattfindet und zielführend für das angestrebte Reflexionsziel ist. (Was hier passiert ist Reflektieren über die Selbstreflexion.)	... konzipieren eigenständig einen Leitfaden zur Selbstreflexion (ggf. gemeinsam mit einem zugehörigen Übungsszenario), der anderen Personen als Anleitung für deren Selbstreflexion über eine bestimmte Situation, Vorgehensweise oder ein selbst erstelltes Artefakt als Leitfaden dient und dabei gezielt auf diejenige Erkenntnis hinarbeitet, die als Ziel der Reflexionsarbeit gewünscht ist.
... identifizieren ein Ziel („Kritikziel“, das sie mit dem Geben bzw. Empfangen von Feedback erreichen möchten. ... überlegen sich in der Rolle als Feedback-Geber, wie die Kritik formuliert werden muss, damit das Gegenüber sie annehmen kann. ... setzen sich in ihrer Rolle als Feedback-Nehmer inhaltlich mit Kritik und Verbesserungsvorschlägen auseinander und versuchen, diese nachzuvollziehen.	... reflektieren in ihrer Rolle als Feedback-Geber ergebnisoffen darüber, welche der von ihnen identifizierten Kritikpunkte im gegebenen Kontext zielführend sind. ... reflektieren in ihrer Rolle als Feedback-Nehmer ergebnisoffen darüber, welche der empfangenen Kritikpunkte und Verbesserungsvorschläge sie für sich als schlüssig und anwendbar erachten.	... konzipieren eigenständig einen Leitfaden und ggf. ein Übungsszenario, das andere Personen als Anleitung für das Geben bzw. Empfangen von Feedback nutzen können, um deren Kritikfähigkeit zu schulen.
... finden einen stringenten Ansatz, um aus einer vorgegebenen Menge von konkreten Beispielen ein passendes Regelwerk zu extrahieren und wählen dazu eine geeignete Abstraktions- bzw. Lösungsstrategie aus. ... wenden zum Lösen einer Abstraktionsaufgabe eine Metastrategie an, um eine geeignete Abstraktions- bzw. Lösungsstrategie zu identifizieren.	... bewerten, ob das Ergebnis einer Abstraktionsaufgabe (d.h. ein abstraktes Modell bzw. Regelwerk) eine für diese Aufgabe sinnvolle Abstraktion ist, inkl. der gewählten Struktur der Abstraktionsebenen. ... prüfen und bewerten, ob das (eigene oder fremde) Vorgehen bei der Abstraktion zielführend und effizient ist.	... entwickeln selbst einen Leitfaden für Abstraktion in einem bestimmten Kontext. ... erarbeiten selbst eine Aufgabe, die die Anwendung abstrakten Denkens erfordert. ... konzipieren zu einem neuem Kontext und einer neuen Art von Aufgabe eigenständig eine (neuartige) Abstraktionsstrategie. ... entwickeln selbst ein neuartiges Metamodell.

im Berufsleben mittelfristig eine Führungstätigkeit an. Folglich ist auch die Kritikfähigkeit in der Rolle der Kritikgebenden eine erforderliche Schlüsselkompetenz.

Da viele unserer Studierenden weder in der gebenden noch in der empfangenden Rolle in ausreichendem Maße über Kritikfähigkeit verfügen, fokussieren wir diesen Bereich im Sinne der Studierfähigkeit als zentrale Sozialkompetenz (siehe Tabellen 6 und 7 (Mitte)). Wünschenswert im Sinne der Studierfähigkeit wäre auch hier eine Kompetenzebene von mindestens Stufe 4 (Analysieren), besser noch Stufe 5 (Evaluieren).

Viele Studierende befinden sich hier jedoch auf Stufe 1 (Erinnern). Beispielsweise ist zu beobachten, dass viele Studierende auch auf angemessen formulierte Verbesserungshinweise mit langen „Ja aber"-artigen Rechtfertigungen oder Ähnlichem reagieren. Ein Grundverständnis dafür, dass Kritik eine wertvolle Unterstützung bei der eigenen Weiterentwicklung darstellen kann, ist in diesen Fällen offensichtlich nicht vorhanden. Feedback-Regeln, die insbesondere in der Rolle des Feedback-Gebers einzuhalten sind, sind vielen Studierenden ebenfalls nicht bekannt.

Methodenkompetenz: Abstraktes Denken

Viele fachliche Inhalte der Informatik befassen sich mit Konzepten, die nicht physisch greifbar sind und somit ein hohes Maß an abstraktem Denken erfordern. Des Weiteren erfordert jeglicher fachliche Kompetenzerwerb, der über rein auswendig gelerntes Wissen hinaus geht, ein Grundmaß an Abstraktion, da bereits für das Verstehen (Stufe 2) eines jeglichen fachlichen Sachverhaltes dessen grundlegende Gesetzmäßigkeiten erfasst werden müssen. Diese grundlegenden Gesetzmäßigkeiten sind selbst bereits eine Abstraktion konkreter Einzelbeobachtungen aus der realen Welt. Entsprechend ist das *abstrakte Denken* eine weitere Schlüsselkompetenz, die insbesondere für informatiknahe Studiengänge von entscheidender Bedeutung ist.

Tabellen 6 und 7 (unten) definieren für das abstrakte Denken Lernziele auf den verschiedenen Kompetenzebenen nach der überarbeiteten Lernzieltaxonomie von Bloom. Auch hier wäre eine Stufe 4 (Analysieren) oder sogar Stufe 5 (Evaluieren) bei den Studierenden wünschenswert. In der Realität kennen viele Studierende zwar relativ schnell die zentralen Grundbegriffe der Abstraktion und verstehen deren Bedeutung für den informatik-nahen Lernprozess. Spätestens bei Stufe 3 (Anwenden) sind jedoch massive Schwierigkeiten zu beobachten, beispielsweise beim Ableiten einer Regel, eines Modells oder einer Klassenstruktur aus konkreten Beispielen.

Angesichts dieser Defizite in den grundlegenden Schlüsselkompetenzen ist es also nicht verwunderlich, dass Studierende Schwierigkeiten haben, diejenigen fachlichen Kompetenzebenen zu erreichen, für die diese Schlüsselkompetenzen zentrale Voraussetzung sind.

Zusammenfassung und Ausblick

Durch die detaillierte Definition der fachlichen Lernziele über die verschiedenen Kompetenzebenen hinweg haben wir aufgeschlüsselt, was eine softwaretechnische Fachkompetenz auf den einzelnen Ebenen der überarbeiteten Lernzieltaxonomie von Bloom jeweils ausmacht. Analog haben wir auch für die relevanten überfachlichen Schlüsselkompetenzen erarbeitet, welcher genauen Fähigkeit diese auf den jeweiligen Kompetenzebenen entsprechen. Dadurch wurden nicht nur die an die Studierenden gestellten fachlichen Anforderungen klarer herausgearbeitet, sondern auch verdeutlicht, welche Schlüsselfertigkeiten in welchem Umfang Voraussetzung für ein erfolgreiches informatik-nahes Studium sind.

Darauf basierend können wir nun gezielt Lehrkonzepte für Interventionen erarbeiten, die zum einen die relevanten Schlüsselkompetenzen bis auf die erforderlichen Ebenen hin aufbauen, und zum anderen auf dieser Grundlage die gewünschten fachlichen Kompetenzen in den Studierenden entwickeln. Dabei ist zu berücksichtigen, dass die Ausgangsbasis (d. h. die Eingangsqualifikation) der Studierenden an den Hochschulen erfahrungsgemäß sehr heterogen ist, sowohl fachlich als auch hinsichtlich der Schlüsselkompetenzen.

Dank

Wir bedanken uns bei Claudia Walter (DiZ Bayern) für den Tipp, bei der Definition von Lernzielen Verben zu verwenden, die ein von außen wahrnehmbares Verhalten beschreiben, weil erst dadurch die definierten Lernziele auch wirklich testbar werden. Des Weiteren danken wir Ingrid Cavalieri (Coach am DiZ Bayern) für die kritische Durchsicht unserer Lernzieldefinitionen.

Das Autorenteam wurde gefördert durch das BMBF Förderkennzeichen 01PL11025 (Projekt "Für die Zukunft gerüstet"), im Programm "Qualitätspakt Lehre".

Literatur

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

[Bloom u. a. 1956] Bloom, B. S. ; Engelhart, M. B. ; Furst, E. J. ; Hill, W. H. ; Krathwohl, D. R.: *Taxonomy of educational objectives: The classification of educational goals*. New York : David McKay Company, 1956

[Böttcher u. a. 2011] Böttcher, Axel ; Thurner, Veronika ; Müller, Gerhard: Kompetenzorientierte Lehre im Software Engineering. In: *SEUH*, 2011, S. 33–39

[Chur 2004] Chur, Dietmar: Schlüsselkompetenzen – Herausforderung für die (Aus-)Bildungsqualität an Hochschulen. In: Stifterverband für die Wissenschaft (Hrsg.): *Schlüsselkompetenzen und Beschäftigungsfähigkeit – Konzepte für die Vermittlung überfachlicher Qualifikationen an Hochschulen*. Essen, Juni 2004, S. 16–19

[Hristova u. a. 2003] Hristova, Maria ; Misra, Ananya ; Rutter, Megan ; Mercuri, Rebecca: Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. In: *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA : ACM, 2003 (SIGCSE '03), S. 153–156

[Humbert 2006] Humbert, Ludger: *Didaktik der Informatik mit praxiserprobtem Unterrichtsmaterial*. 2. B.G. Teubner, 2006

[Kaczmarczyk u. a. 2010] Kaczmarczyk, Lisa C. ; Petrick, Elizabeth R. ; East, J. P. ; Herman, Geoffrey L.: Identifying Student Misconceptions of Programming. In: *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. New York, NY, USA : ACM, 2010 (SIGCSE '10), S. 107–111

[Knuth 1989] Knuth, Donald E.: The errors of TEX. In: *Software: Practice and Experience* 19 (1989), Nr. 7, S. 607–685

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

[Schaper u. a. 2012] Schaper, Niclas ; Reis, Oliver ; Wildt, Johannes ; Horvath, Eva ; Bender, Elena: *Fachgutachten zur Kompetenzorientierung in Studium und Lehre*. Hochschulrektorenkonferenz, Projekt nexus, 2012

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

[Sirkiä u. Sorva 2012] Sirkiä, Teemu ; Sorva, Juha: Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises. In: *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*. New York, NY, USA : ACM, 2012 (Koli Calling '12), S. 19–28

[In der Smitten u. Jaeger 2009] Smitten, Susanne In d. ; Jaeger, Michael: Kompetenzerwerb von Studierenden und Profilbildung an Hochschulen. In: *HIS-Tagung 2009 – Studentischer Kompetenzerwerb im Kontext von Hochschulsteuerung und Profilbildung*. Hannover : HIS, 2009, S. 1–26

[Sorva 2008] Sorva, Juha: The Same but Different – Students' Understandings of Primitive and Object Variables. In: *Proceedings of the 8th International Conference on Computing Education Research*. New York, NY, USA : ACM, 2008 (Koli '08), S. 5–15

[Zehetmeier u. a. 2015] Zehetmeier, Daniela ; Böttcher, Axel ; Brüggemann-Klein, Anne ; Thurner, Veronika: Development of a Classification Scheme for Errors Observed in the Process of Computer Science Education. In: *Submitted to: Joint Software Engineering Education and Training (JSEET)*, 2015

Übung macht den Meister? Lernaufgabentypen im Hochschulfach Software Engineering

Paula Figas, Alexander Bartel, Georg Hagel, Hochschule Kempten

{paula.figas, alexander.bartel, georg.hagel}@hs-kempten.de

Zusammenfassung

Lernaufgaben spielen in der Hochschuldidaktik, insbesondere im Kontext vorlesungsbegleitender Übungen, eine große Rolle. Umso erstaunlicher ist es, dass es bis dato kaum Untersuchungen in diesem Bereich gibt. Dabei stellen sich zahlreiche Fragen: Wie sind Lernaufgaben in dem tertiären Bildungssektor aktuell konzipiert? Welche Lernaufgabentypen können unterschieden werden? Im Rahmen des Verbundprojekts EVELIN (Abke et al., 2012) wurde eine Dokumentenanalyse von insgesamt 350 Teilaufgaben von fünf deutschen Hochschulen und Universitäten im Fach Software Engineering durchgeführt. Dabei erwiesen sich die vier Aufgabentypen a) Erarbeitungsaufgaben, b) Wiederholungs- und Übungsaufgaben, c) Anwendungsaufgaben und d) freie Gestaltungsaufgaben als besonders wichtig.

Lernaufgaben

Was sind Lernaufgaben?

Was ist eine Lernaufgabe? Klammert man den Lernaspekt zunächst aus und betrachtet lediglich das Substantiv Aufgabe, so wird deutlich, dass dieses verschiedene Bedeutungen vereint: Laut dem Duden (2013) wird es einerseits definiert als „etwas, was jemandem zu tun aufgegeben ist“ und gleichzeitig in Verbindung gesetzt mit dem Aufgeben, im Sinne von Nichtfortsetzen. Im pädagogischen Kontext existiert eine Pluralität an unterschiedlichen Begriffsdefinitionen. Ellis (2003, S.2) weist darauf hin, „…that in neither research nor language pedagogy is there complete agreement as to what constitutes a task”. Während Seel (1981) und Prabhu (1987) Lernaufgaben als ein Instrument der Instruktionspsychologie betrachten und sich Astleitner & Herber (2007) auf kognitivistische Lerntheorien stützen, greifen Krogoll (1998) und Keller & Bender (2012) zentrale Aspekte der Ermöglichungsdidaktik auf, um Lernaufgaben zu beschreiben. Allen Ansätzen gemein ist, dass das *Lernen* im Vordergrund steht: Durch die Aufgabenbearbeitung sollen Lernprozesse angestoßen und eine Kompetenzentwicklung angeregt werden. Nach Keller & Bender (2012, S. 8) können dies schriftliche „...Problemstellungen und Arbeitsanleitungen [sein], welche (...) zur Auseinandersetzung mit einem speziellen Unterrichtsinhalt anregen“. Dabei geht es darum, Wissen aufzubauen und zu vertiefen (Maier, Kleinknecht & Metz, 2010) und darum, „gestaltungs- und kompetenzorientierte Lernsituationen“ bereitzustellen (Bloemen & Schlömer, 2012, S. 128). Dies kann beispielsweise in Form von Aufforderungen geschehen, bestimmte Handlungen auszuführen, Fragen zu beantworten oder Probleme zu lösen (Pahl, 1998). Lernaufgaben können auch spielerisch – beispielsweise durch einen Gamification-basierten Ansatz – Eingang in die Lehre finden (Bartel, Figas & Hagel, 2014a).

Lernaufgaben sind zu separieren von lerndiagnostischen Aufgaben, wie Test- und Leistungsaufgaben: Testaufgaben zielen auf das Überprüfen und Diagnostizieren von Wissen und Kompetenzen ab (Maier, Kleinknecht & Metz, 2010). Leistungsaufgaben dienen der Überprüfung des erreichten Kompetenzniveaus und werden daher häufig benotet. Hierbei stehen Leistungsziele und Erfolge im Vordergrund (Abraham & Müller, 2009).

Eine Lernaufgabe bewegt sich stets im Spannungsfeld der didaktischen Fragen: *Wer* (Zielgruppe) soll *was* (Inhalt) von *wem* (Lehrende) *wann* (Zeitpunkte) mit *wem* (Sozialform) *wo* (Lernorte) *wie* (Methoden) *womit* (Medien) *wozu* (Zielsetzung) lernen (Frank & Iller, 2013)? Dementsprechend viele fachdidaktische Ansätze gibt es, Lernaufgaben näher zu kategorisieren und zu differenzieren. Arnold & Thillosen (2002) ziehen als Unterscheidungsmerkmale unter anderem die Position im Lernmaterial, die Sozialform der Aufgabenbearbei-

tung, den Komplexitätsgrad oder die angestrebten Kompetenzen heran.

Die Funktion von Lernaufgaben

Nachdem Lernaufgaben in der Forschung lange Zeit verhältnismäßig wenig beachtet blieben, wurden sie im Zuge der kompetenzorientierten Schulpädagogik wieder neu entdeckt und rückten stärker in den Fokus von Publikationen und wissenschaftlichen Diskursen (Matthes & Schütze, 2011). Immer mehr Ansätze beschäftigen sich mit der Konstruktion von kompetenzorientierten Lernaufgaben und betonen dabei die zentrale didaktische Funktion der Aufgaben: Mittels Lernaufgaben können Lehrende und Lernende miteinander in Kommunikation treten, wodurch sie „...zu Mittlern zwischen beiden ‚Welten' werden" (Matthes & Schütze 2011, S. 10). Oelkers & Reusser (2008, S. 408) bezeichnen Lernaufgaben sogar als „Rückgrat des Unterrichts":

> *„Gute fachliche Lernaufgaben materialisieren jene Wissens- und Könnenskomponenten, lösen jene Denk- und Arbeitsprozesse aus und aktivieren jene analytischen und synthetischen Figuren des Problemlösens, Argumentierens, Betrachtens und Deutens, um die es in einem bestimmten Fach im Kern geht und die dessen intellektuelle Kultur ausmachen" (Oelkers & Reusser, 2008, S. 408).*

Eine Besonderheit von Lernaufgaben besteht darin, dass sie auf verschiedene Arten gestellt werden können, wodurch eine gewisse Steuerung seitens der Lehrenden möglich ist. Die Bearbeitung kann hingegen unabhängig und selbstständig erfolgen. Lernmöglichkeiten ergeben sich im umfassenden Lernaufgabenprozess nicht nur während und in der Bearbeitungsphase, sondern ebenso in der thematischen Einführung der Aufgabenstellung durch die Lehrperson, der (gemeinsamen) Besprechung oder dem abschließenden Feedback zu den individuellen Lösungen (Figas et al., 2014).

Dementsprechend viele didaktische Funktionen können den Lernaufgaben zugeschrieben werden: Sie unterstützen bei der Beantwortung der pädagogischen Grundfragen, wie und was gelehrt werden soll und dienen der Repräsentation und Veranschaulichung von Wissensinhalten. Darüber hinaus geben sie die Möglichkeit das Gelernte zu wiederholen, zu üben und anzuwenden. Es können Reflektions- und Denkprozesse angestoßen und die Lernenden dazu ermutigt werden, eigene Lösungsmodelle zu entwickeln (z.B. Steindorf, 2000; Heitzmann & Niggli, 2010).

Lernaufgaben im Hochschulfach Software Engineering (SE)

Neuere Ansätze der Lernaufgabenkonstruktion beziehen sich häufig auf den schulpädagogischen Bereich. Doch auch im Hochschulkontext spielen sie in vielen Studiengängen eine zentrale Rolle. Dies ist beispielsweise bei zahlreichen Veranstaltungen zum Thema Software Engineering (SE) der Fall (Figas & Hagel 2014). Software Engineering – auch Softwaretechnik genannt – ist eine technische Disziplin, welche „…sich mit allen Aspekten der Softwareherstellung beschäftigt, von den frühen Phasen der Systemspezifikation bis hin zur Wartung des Systems" (Sommerville, 2012, S. 33). Ludewig & Lichter (2007) verstehen darunter alle Aktivitäten, welche die Erstellung oder Veränderung von Software beinhalten.

In nahezu allen informatiknahen Studiengängen gehört das Fach Software Engineering zum festen Bestandteil der hochschulischen Ausbildung (z.B. Otto-Friedrich Universität Bamberg, 2013; Technische Universität Darmstadt, 2007). In einigen Universitäten und Hochschulen gibt es Softwaretechnik sogar als eigenen Studiengang (z.B. Universität Stuttgart, 2012; Fachhochschule Dortmund, 2014). Viele Hochschulveranstaltungen in dieser Disziplin sind zweigeteilt: Eine Vorlesung wird beispielsweise von regelmäßig stattfindenden Übungen, Tutorien oder Praktika begleitet, in welchen die Inhalte der Vorlesung gemeinsam mit Übungsleitern vertieft betrachtet werden. Dies geschieht primär über Aufgaben, welche die Studierenden bearbeiten und mit Übungsleitern besprechen. Insofern rücken Lernaufgaben hier in ein besonderes Licht, da sie in vielen Fällen zum regulären Bestandteil der Veranstaltungen gehören. Doch wie sehen Lernaufgaben in diesem Fach aus? Welche Lernaufgabentypen gibt es hier und in welcher Häufigkeit tauchen sie auf? Um Antworten auf diese Fragen zu finden, wurde eine systematische Dokumentenanalyse durchgeführt.

Eine Dokumentenanalyse von Lernaufgaben im Hochschulfach Software Engineering

Methodik und Vorgehen

Zentrales Ziel der Untersuchung ist es herauszufinden, welche Lernaufgabentypen im Hochschulfach Software Engineering unterschieden werden können, wie diese konkret aussehen und welche Rückschlüsse dies für die Lehre von Software Engineering zulässt. Dafür wurde eine systematische Dokumentenanalyse durchgeführt. Nach Ballstaedt (1987, S. 203) geht es in einer Dokumentenanalyse um die Interpretation und Analyse von „...Hervorbringungen oder Zeugnisse[n] menschlichen Handelns, Denkens und Erlebens, die in

natürlichen Situationen entstanden sind und erst nachträglich zur Beantwortung einer Forschungsfrage herangezogen werden". Fokussiert werden demnach vorhandene Materialien, welche nicht mehr erhoben werden müssen. Dies können sowohl Filme, Tonbänder, Gegenstände, Texte als auch – wie in vorliegender Studie – didaktisches Material, wie Lernaufgaben, sei (Ballstaedt, 1987).

In der Dokumentenanalyse wurden Aufgaben von Lehrveranstaltungen mit dem Namen „Softwaretechnik" bzw. „Software Engineering" aus fünf zufällig ausgewählten deutschen Hochschulen untersucht. Insgesamt beträgt die Stichprobe, inklusive aller Teilaufgaben, N = 350 Aufgaben. Unter Teilaufgaben werden dabei Unterpunkte einer Aufgabe verstanden, welche in sich abgeschlossene Arbeitsanweisungen darstellen. Häufig werden diese durch Aufzählungszeichen oder Nummerierungen voneinander getrennt und signalisieren damit einen thematischen Zusammenhang zwischen den Aufgaben einerseits und eine eigenständige Aufgabenlogik andererseits. Sie sind jeweils eingebettet in Aufgabenblätter, welche die Studierenden begleitend zur Vorlesung wöchentlich bearbeiten und im Anschluss in Übungen oder Tutorien besprechen.

Im Schnitt befinden sich 6,03 Aufgaben auf einem Aufgabenblatt. Die Dokumente selbst sind in sich vollständig und liegen in digitaler textueller Form vor. Der inhaltliche Schwerpunkt liegt dabei auf folgenden Bereichen:

- *Softwareentwurf:* Hierbei liegt der Fokus besonders auf dem Einsatz der UML (Unified Modeling Language) als Modellierungssprache mit ihren gängigen Diagrammtypen (Anwendungsfall-, Aktivitäts-, Zustands-, Komponenten-, Objekt- und Klassendiagramm).
- *Programmierung*: Zahlreiche Aufgaben adressieren die Umsetzung objektorientierter Konzepte oder Entwurfsmuster mit den Programmiersprachen Java, C# oder C++. Vereinzelt sollen Aufgaben auch mit Pseudocode bearbeitet werden.
- *Entwicklungswerkzeuge und -methoden:* Für die Bearbeitung von Aufgaben werden UML-Modellierungswerkzeuge, wie der Sparx Enterprise Architect oder integrierte Entwicklungsumgebungen, wie Eclipse oder Visual Studio, eingesetzt.
- *Projektmanagement:* Wesentliche Inhalte von Aufgaben sind hier vor allem die Anwendung von einschlägigen Methoden und Techniken des Projektmanagements, wie die Anfertigung einer Software-Risikoanalyse.
- *Softwaretest:* Hierbei geht es insbesondere um Testprozesse und dazugehörige Testarten.

Insgesamt ist eine große Heterogenität bezogen auf die Inhalte der einzelnen Aufgaben an unterschiedlichen Hochschulen zu verzeichnen. In jeder Lehrveranstaltung sind thematische Schwerpunkte zu erkennen, trotz der Vielfalt an Software Engineering Themen.

Ergebnisse – Lernaufgabentypen

Die Stichprobe wurde auf Merkmale unterschiedlicher Aufgabentypen untersucht. Entscheidend dabei war die Aufgabenintention: Wozu werden die Studierenden angehalten? Sollen sie die Lösung selbst erarbeiten oder etwas bewerten? Sollen sie etwas selbst entwerfen? Dabei wurde sich an bestehenden Ansätzen der didaktischen Aufgabendifferenzierung orientiert (z.B. Hinney et al., 2008; Matthes & Schütze, 2011).

Die Ergebnisse der Analyse zeigen ein buntes Bild: Viele verschiedene Formate finden in Veranstaltungen zum Thema Software Engineering Eingang. Sowohl vom Inhalt, der Instruktion, der grafischen Gestaltung als auch den vorgegebenen Rahmenbedingungen unterscheiden sich die Aufgaben zwischen den Hochschulen und sogar innerhalb einer Veranstaltung.

6 Prozent der untersuchten Aufgaben enthalten eine Aufforderung, sich selbst Wissensinhalte zu erarbeiten. Ein verhältnismäßig häufiges Format hierfür ist das Thematisieren eines bis dato unbekannten Inhaltes und die daran geknüpfte Aufforderung, Begriffe oder den Sachverhalt selbst in der Literatur nachzuschlagen. Diese Aufgaben können unter dem Terminus *Erarbeitungsaufgaben* subsummiert werden (vgl. Abb. 1). Sie dienen im Lernprozess der Erarbeitung eines Themas bzw. Gegen-

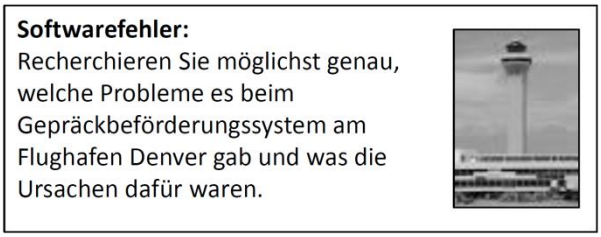


Abb. 1: Beispiel für eine Erarbeitungsaufgabe

standbereichs. Hierbei soll die „Aneignung und Ausdifferenzierung subjektiver Regelhypothesen und metasprachlicher Prüfoperationen bei den Lernenden" erfolgen (Hinney et al., 2008).

38 Prozent der Aufgaben zielen darauf ab, bereits gelernte Inhalte noch einmal abzufragen, zu festigen oder zu wiederholen. Dabei sollen Fachbegriffe definiert und Beispiele dazu gegeben werden. Es geht darum, bereits bekannte Sachverhalte in eigenen Worten zu umschreiben, individuelle Beispiele zu finden oder Zusammenhänge zwischen Themenbereichen herzustellen. An manchen Hochschulen gehört eine fünfminütige Präsentation zum festen Bestanteil jeder Übung, in welcher je fünf subjektiv als wichtig

empfundene Punkte der letzten Vorlesung von den Studierenden zusammengefasst, erläutert und mit den Kommilitonen diskutiert werden. Diese Art der

Begriffsbestimmung:
Definieren Sie drei Begriffe, welche Sie in der Vorlesung kennengelernt haben:

Begriff	Definition	Beispiel
Assoziation		
Aggregation		
Komposition		

Was genau bedeuten diese Begriffe? Geben Sie für jede Definition ein anschauliches Beispiel an!

Abb. 2: Beispiel für eine Wiederholungs- und Übungsaufgabe

Aufgaben wurde zu *Wiederholungs- und Übungsaufgaben* zusammengefasst (vgl. Abb. 2). Sie fungieren als Instrument, Gelerntes zu wiederholen und zu vertiefen (Hinney et al., 2008).

Rund die Hälfte der Aufgaben, 51 Prozent, fokussiert die praktische Anwendung von bereits gelernten Inhalten. In den meisten Fällen handelt es sich dabei um das Modellieren von UML-Diagrammen. In verschiedenen Konstellationen werden die Studierenden dazu aufgefordert, zu fiktiven oder realistischen Beispielen und Szenarien eigene

Modellierung eines Klassendiagramms:
Entwerfen Sie ein vollständiges Klassendiagramm mit der folgenden Beschreibung. Wenden Sie – wenn möglich – Entwurfsmuster an, welche Sie in der Vorlesung kennengelernt haben. Bitte treffen Sie sinnvolle Annahmen, sofern Ihnen Angaben aus dem Text fehlen.

Ein Zeichenprogramm, was Sie entwerfen sollen, muss in der Lage sein, grafische Objekte zu zeichnen. Dies geschieht auf einer Zeichenfläche, deren Größe einem standardisierten DIN-Format folgt und welches zwei Arten von Hintergrundmustern darstellen kann: linierte und karierte. Das Zeichenprogramm kann bis zu 10 Zeichenflächen anzeigen. Es gibt drei Arten von grafischen Objekten: Punkte, Strecken und Polygone. Beliebig viele grafische Objekte können auf der Zeichenfläche dargestellt und miteinander zu Figuren zusammengefasst werden. Figuren sollen als wiederverwendbare Templates gespeichert werden können. Zur Darstellung besitzen Punkte eine x- bzw. y-Koordinate. Strecken bestehen aus Punkten und Polygone aus Strecken.

Abb. 3: Beispiel für eine Anwendungsaufgabe

Diagramme nach den vorgegebenen Richtlinien der UML zu erstellen und somit das theoretische Gerüst aus der Vorlesung selbst praktisch zu erproben (vgl. Abb. 3). An manchen Universitäten werden die Aufgaben dazu in ein Beispielprojekt eingebettet, welches die Studierenden ein ganzes Semester begleitet und anhand dessen verschiedene UML-Diagramme geübt werden können (Bartel, Figas & Hagel, 2014b). Auch einfache Programmieraufgaben werden dazu gezählt. Diese Merkmale entsprechen dem Lernaufgabentyp *Anwendungsaufgabe*. Diese intendieren, das Gelernte auf neue Situationen zu übertragen. Dazu werden unter anderem Analyse-, Transfer-, Problemlösungs- und Strategieaufgaben gezählt (Matthes & Schütze, 2011, S. 10).

Nur wenige Aufgaben (5 Prozent) enthalten eine Aufforderung, etwas Eigenes, Neues zu entwickeln. Hierbei geht es darum, eigene Ideen zu generieren. Meist bezieht sich dies auf Modellierungsaufgaben

Eigene Idee für ein Softwareprojekt:
Entwickeln Sie eine eigene realistische Idee für eine zu entwickelnde Software.

- Bei Ihrer Idee soll es sich um eine relativ einfache, nicht zu komplexe Software handeln. Es soll sich bspw. nicht um ein Computerspiel handeln.
- Die Software sollte Sie thematisch interessieren und/oder Sie sollten einen beruflichen oder privaten (Hobby, Verein etc.) Einsatzzweck sehen.
- Es kann sich bei dieser Software um eine Idee handeln mit der Sie sich schon in irgendeiner Form beschäftigt haben, jedoch darf es sich nicht um ein fertig existierendes Softwareprojekt handeln.

Abb. 4: Beispiel für eine freie Gestaltungsaufgabe

oder komplexe Programmieraufgaben (vgl. Abb. 4). Dieser Lernaufgabentypus zählt zu den *freien Gestaltungsaufgaben*. Dabei geht es darum, „…eine Situation zu planen und zu realisieren, ein Verfahren zu entwerfen oder ein Produkt zu konzipieren“ (Tulodziecki, Herzig & Blömeke, 2004, S.94). Im Gegensatz zu den anderen Lernaufgabentypen sind hierbei sowohl Kreativität als auch fundierte fachliche Kenntnisse Voraussetzung zur Bewältigung der Aufgaben.

Rückschlüsse für Lernaufgaben in der kompetenzorientierten SE-Lehre

Insgesamt konnten vier Lernaufgabentypen in den untersuchten Dokumenten identifiziert werden. Andere, in der Literatur ebenso als bedeutsam beschriebene Lernaufgabentypen, wie Beurteilungsaufgaben (Tulodziecki, Herzig & Blömeke, 2004), konnten in den analysierten Aufgaben nicht wiedergefunden werden. Dies zeigt bereits deutlich, worauf der Fokus der Aufgaben im Bereich der hochschulischen Software Engineering Ausbildung insgesamt liegt: Bereits in der Vorlesung gehörte Inhalte sollen in einer tieferen Ebene durchdrungen, ergänzt, vertieft und praktisch angewendet werden.

Betrachtet man die Ergebnisse der Studie unter der Lupe einer durch Bologna geforderten kompe-

tenzorientierten Didaktik, so lässt dies Rückschlüsse auf die Lehre von Software Engineering zu. Kompetenzen sind weit mehr als reines Fachwissen. Sie umfassen die Fähigkeiten, Fertigkeiten und das Wissen einer Person, bestimmte Probleme zu lösen sowie die damit verbundene Bereitschaft die Problemlösungen kreativ und selbstorganisiert in neuen Situationen nutzen zu können (Erpenbeck & Rosenstiel, 2007; Weinert, 2001). Insbesondere in der Softwareentwicklung erfordern viele Aktivitäten ein hohes Maß an Selbstorganisation und Kreativität, wie beispielsweise das Modellieren von Fachkonzepten, das Entwerfen von Systemarchitekturen oder das Suchen von neuen Algorithmen (Balzert, 2008).

Wie in Tab. 1 zusammengefasst, ergeben sich hierbei Unterschiede bei den jeweiligen Aufgabentypen: Bei Erarbeitungs- sowie Übungs- und Wiederholungsaufgaben soll die Bearbeitung unter Bezugnahme von Lösungsrichtlinien zu einem definierten Ergebnis führen und bei den Studierenden in erster Linie zu einem Wissenserwerb beitragen. Dabei ist der Gestaltungsspielraum einer Lösung bei der Bearbeitung dieses Aufgabentyps durch den fachlichen Inhalt eingegrenzt. Die Gruppe der Anwendungsaufgaben stellt den weitverbreitetsten Lernaufgabentyp im Bereich Software Engineering dar. Auch hierbei geht es darum, unter Einbezug von bestehendem Wissen Inhalte angeleitet und selbstständig anwenden zu können. Bei den freien Gestaltungsaufgaben hingegen sollen Lernende mittels Fachwissen selbstständig etwas Eigenes entwickeln und zuvor unbekannte Probleme lösen. Daran wird ersichtlich, dass für eine nachhaltige Kompetenzentwicklung insbesondere die freien Gestaltungsaufgaben von Bedeutung sind. Gleichzeitig legen die Ergebnisse der Analyse nahe, dass dieser Aufgabentyp nur sehr sporadisch in der Lehre eingesetzt wird und dass damit nur selten über die reine Wissensanwendung hinausgegangen wird.

In diesem Zusammenhang ist darauf hinzuweisen, dass die Lernaufgabentypen aufeinander aufbauen und in einem gestaffelten Lehrkonzept Eingang in die Hochschuldidaktik finden. So müssen Studierende beispielsweise in der Lage sein, ein UML-Klassendiagramm in einem abgegrenzten Kontext selbstständig zu erstellen. Das hierfür benötigte Wissen, z.B. die entsprechenden Notationselemente, kann durch Erarbeitungs- sowie Übungs- und Wiederholungsaufgaben erlangt und gefestigt und in einem späteren Schritt in Anwendungs- und Gestaltungsaufgaben vertieft werden. Denn wie bereits Romeike (2007, S. 61) feststellte, ist„…ein solides Grundwissen im Betätigungsgebiet Voraussetzung für jeden kreativen Prozess“. Dies deutet darauf hin, dass vermittelte Kenntnisse, Arbeitspraktiken oder -routinen als Fundament für die Kompetenzentwicklung fungieren können, jedoch darauf aufbauend neue Situationen in der Lehre geschaffen werden müssen, die es erfordern, selbstorganisiert und kreativ zu arbeiten. Daher rückt die Frage in den Vordergrund, wie freie Gestaltungsaufgaben verstärkt in die Hochschullehre implementiert werden können ohne dass der Arbeitsaufwand für die Dozierenden zu sehr ansteigt. Möglichkeiten hierfür ergeben sich beispielsweise in der Einbettung neuer Medien und der damit verbundenen Lernformen (E-/M-Learning) in den Vorlesungsbetrieb. So können in einer Lehrveranstaltung Wiederholungsfragen zu bereits erlerntem Stoff onlinebasiert zur Verfügung gestellt werden, um nicht nur das Faktenwissen von Studierenden repetitiv zu erweitern, sondern zudem Abwechslung einzubringen und Selbstbestimmung im Lernprozess zu fördern (Bartel, Figas & Hagel, 2014b). Damit können die Aneignung und Wiederholung von Wissen zuzüglich zur Vorlesung systematisch unterstützt und Kapazitäten dafür

	Erarbeitungs-aufgaben	Wiederholungs- und Übungsaufgaben	Anwendungs-aufgaben	Freie Gestaltungs-aufgaben
Didaktische Funktion	Themen kennenlernen, eigene Meinung bilden und reflektieren.	Themen vertiefen und festigen.	Inhalte angeleitet und selbstständig anwenden.	Mit inhaltlichem Wissen etwas Eigenes schaffen.
Voraussetzung	Es wird wenig oder kein Wissen vorausgesetzt.	Wissen wird vorausgesetzt.	Fundiertes Wissen wird vorausgesetzt.	Fundiertes Wissen wird vorausgesetzt.
Antwortformat	Tendenziell wenige unterschiedliche Antwortmöglichkeiten, relativ genaue Musterlösungen möglich.	Tendenziell wenige unterschiedliche Antwortmöglichkeiten, relativ genaue Musterlösungen möglich.	Viele Antwortmöglichkeiten, Richtlinien für richtige und falsche Ansätze.	Sehr viele Antwortmöglichkeiten, grobe Richtlinien für geeignte Ansätze.
Grad der Offenheit	Gering	Eher gering	Eher hoch	Sehr hoch

Tab. 1: Übersicht der Lernaufgabentypen

geschaffen werden, Anwendungsaufgaben zu freien Gestaltungsaufgaben weiter zu entwickeln.
Für die Studierenden gilt es, einen geeigneten zeitlichen Rahmen für die Aufgabenbearbeitung als Orientierung zu setzen. Für Dozierende hingegen lässt sich dem Problem des höheren Korrektur- und Besprechungsaufwands von freien Gestaltungsaufgaben nur durch mehr zeitliche oder personelle Ressourcen beikommen. Denn selbst vielversprechende Verfahren, wie die automatische Konsistenzprüfung von UML-Diagrammen (z.B. Rasch & Wehrheim, 2003), lassen nur eingeschränkt eine Aussage über die Qualität einer Lösung zu einer freien Gestaltungsaufgabe – in diesem Fall einer Modellierungsaufgabe – zu.

Zusammenfassung

Es gilt festzuhalten, dass Lernaufgaben im Fach Software Engineering in den Übungen eine bedeutende Rolle einnehmen. Das typische Format besteht dabei aus Aufgabenblättern, welche eine Zusammenstellung von unterschiedlichen Lernaufgaben beinhalten.

Insgesamt lassen sich vier Typen von Lernaufgaben unterscheiden, welche je eine unterschiedliche Bedeutung zu haben scheinen: Der Fokus der Lernaufgabentypen liegt auf Anwendungsaufgaben. Diese ermöglichen es, den theoretischen Input der Vorlesungen durch praktische Anwendung anzureichern. In allen fünf untersuchten Bildungseinrichtungen wurde dies bestätigt. Wiederholungs- und Übungsaufgaben nehmen ebenfalls eine sehr bedeutsame Rolle ein, Erarbeitungsaufgaben werden hingegen fast gar nicht eingesetzt. Möglicherweise ist die Ursache dafür, dass die Aufgaben als Ergänzung zur Vorlesung konzipiert werden. Auch freie Gestaltungsaufgaben wurden insgesamt nur sehr sporadisch festgestellt, wenngleich in ihnen das größte Potenzial liegt, dem Anspruch einer kompetenzorientierten Lehre gerecht zu werden.

Danksagungen

Die vorliegende Arbeit wird unter dem Förderkennzeichen 01PL12022C durch das vom BMBF finanzierte Verbundprojekt „Experimentelle Verbesserung des Lernens von Software Engineering“ (EVELIN) gefördert.

Literatur

Abke, J. et al. (2012): EVELIN. Ein Forschungsprojekt zur systematischen Verbesserung des Lernens von Software Engineering. In: Embedded Software Engineering Kongress. Sindelfingen, 653–658.

Abraham, U. / Müller, A. (2009): Aus Leistungsaufgaben lernen. Praxis Deutsch, 36, 214, 4–12.

Arnold, P. / Thillosen, A. (2002): Aufgabenorientiertes Lernen in telematischen Studienmodulen: Aufgabenformen, Aufgabentypen und Aufgabengestaltung. In: Zimmer, G. (Hrsg.): High-Tech or High-Teach: Lernen in Netzen zwischen Aktualität und Potenzialität. Dokumentation der Beiträge im Workshop 7 der Hochschultage Berufliche Bildung 2002 an der Universität zu Köln. Bielefeld: Bertelsmann, 35–45.

Astleitner, H. / Herber, H.-J. (Hrsg.) (2007): Task- and Standard-based Learning. Frankfurt: Peter Lang.

Ballstaedt, S. P. (1987): Zur Dokumentenanalyse in der biographischen Forschung. In: Jüttemann, G. / Thomae, H. (Hrsg.): Biographie und Psychologie. Berlin und New York: Springer, 203–214.

Balzert, H. (2008): Lehrbuch der Softwaretechnik. Softwaremanagement. Heidelberg: Spektrum.

Bartel, A. / Figas, P. / Hagel, G. (2014a): Mobile Game-Based Learning in University Education. In: Feller, S. / Yengin, I. (Hrsg.): Educating in Dialog: Constructing meaning and building knowledge with dialogic technology. Amsterdam: Benjamins, 159–180.

Bartel, A. / Figas, P. / Hagel, G. (2014b): Using a Scenario-Based Approach for Learning Software Engineering. In: Hagel, G. / Mottok, J. (Hrsg.): Proceedings of the European Conference on Software Engineering Education. Aachen: Shaker Verlag, 167–179.

Bloemen, A. / Schlömer, T. (2012): Berufliche Handlungskompetenz. In: Paechter, M. et al. (Hrsg.): Handbuch Kompetenzorientierter Unterricht. Weinheim und Basel: Beltz, 119–139.

Duden (2013): Duden online. Deutsche Rechtschreibung.URL: http://www.duden.de/rechtschreibung/Aufgabe, Zugriff am 11.09.2013.

Ellis, R. (2003): Task-based Language Learning and Teaching. New York: Oxford University Press.

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

Fachhochschule Dortmund 2014: Softwaretechnik dual. URL: http://www.fh-dortmund.de/de/fb/4/lehre/swt_dual/index.php, Zugriff am 24.09.2014.

Figas, P. et al. 2014: Man wächst mit seinen Aufgaben: Über die kompetenzorientierte Konstruktion von Lernaufgaben in der Hochschullehre am Beispiel von Software Engineering: In: Ralle, B. et al. (Hrsg.): Lernaufgaben entwickeln, bearbeiten und überprüfen. Ergebnisse und Per-

spektiven der fachdidaktischen Forschung. Münster: Waxmann, 246–249.

Figas, P. / Hagel, G. (2014): Fostering Creativity of Software Engineers through Instructional Tasks? In: Hagel, G. / Mottok, J. (Hrsg.): Proceedings of the European Conference on Software Engineering Education. Aachen: Shaker Verlag, 31–44.

Frank, S. / Iller, C. (2013): Kompetenzorientierung – mehr als ein didaktisches Prinzip. In: Report. Zeitschrift für Weiterbildungsforschung. 36, 4, 33–41.

Heitzmann, A. / Niggli, A. (2010): Lehrmittel: Ihre Bedeutung für Bildungsprozesse und die Lehrerbildung. Beiträge zur Lehrerbildung, 28, 1, 6–19.

Hinney, G. et al. (2008): Definition und Messung von Rechtschreibkompetenz. Didaktik Deutsch. Sonderheft, 2, 107–126.

Keller, S. / Bender, U. (2012): Aufgabenkulturen. Fachliche Prozesse herausfordern, begleiten, reflektieren. Seelze: Kallmeyer und Klett.

Krogoll, T. (1998): Lernaufgaben: Gestalten von Lernen und Arbeiten. In: Holz, H. (Hrsg.): Lern- und Arbeitsaufgabenkonzepte in Theorie und Praxis. Bielefeld: Bertelsmann, 148–164.

Ludewig, J. / Lichter, H. (2007): Software Engineering: Grundlagen, Menschen, Prozesse, Techniken. Heidelberg: dpunkt-Verlag.

Maier, U. / Kleinknecht, M. / Metz, K. (2010): Ein fächerübergreifendes Kategoriensystem zur Analyse und Konstruktion von Aufgaben. In: Kiper, H. et al. (Hrsg.): Lernaufgaben und Lernmaterialien im kompetenzorientierten Unterricht. Stuttgart: Kohlhammer, 28–43.

Matthes, E. / Schütze, S. (2011): Aufgaben im Schulbuch. Einleitung. In: Matthes, E. / Schütze, S. (Hrsg.): Aufgaben im Schulbuch. Bad Heilbrunn: Klinkhardt, Beiträge zur historischen und systematischen Schulbuchforschung, 9–15.

Oelkers, J. / Reusser, K. (2008): Qualität entwickeln - Standards sichern - mit Differenzen umgehen. In: Bundesministerium für Bildung und Forschung (Hrsg.): Bildungsforschung. Band 27, Bonn und Berlin.

Otto-Friedrich-Universität Bamberg (2013): Modulhandbuch: Bachelorstudiengang Wirtschaftsinformatik. Fakultät Wirtschaftsinformatik und Informatik. URL: http://www.uni-bamberg.de/fileadmin/beauftragter_ba_wi.wiai/MHB_BaWI_WS1112.pdf, Zugriff am 10.07.2014.

Pahl, J. P. (1998): Berufsdidaktische Perspektiven der Lern- und Arbeitsaufgaben. In: Holz, H. (Hrsg.): Lern- und Arbeitsaufgabenkonzepte in Theorie und Praxis. Bielefeld: Bertelsmann, 13–30.

Prabhu, N. S. (1987): Second Language Pedagogy. New York: Oxford University Press.

Rasch, H. / Wehrheim, H. (2003): Checking Consistency in UML Diagrams: Classes and State Machines. In: Najm, E. / Nestmann, U. / Stevens, P. (Hrsg.): Formal Methods for Open Object-Based Distributed Systems. Paris: Springer, 229–243.

Romeike, R. (2007): Kriterien kreativen Informatikunterrichts. In: Schubert, S. (Hrsg.): Didaktik der Informatik in Theorie und Praxis. 12. GI-Fachtagung Informatik und Schule. Bonn, 57–68.

Seel, N. M. (1981): Lernaufgaben und Lernprozesse. Stuttgart u.a: Kohlhammer.

Sommerville, I. (2012): Software Engineering. 9. Auflage. München: Pearson.

Steindorf, G. (2000): Grundbegriffe des Lehrens und Lernens. 5. Auflage. Bad Heilbrunn: Klinkhardt.

Technische Universität Darmstadt (2007): Modulhandbuch für den Bachelor- / Masterstudiengang Informatik. URL: http://www.ist.tu-darmstadt.de/media/ist/ordnungen/po2007/modulhandbuch_mit_lehrveranstaltungen_des_fb20.pdf, Zugriff am 10.07.2014.

Tulodziecki, G. / Herzig, B. / Blömeke, S. (2004): Gestaltung von Unterricht: Eine Einführung in die Didaktik. Bad Heilbrunn: Klinkhardt.

Universität Stuttgart (2012): Softwaretechnik. Abrufbar unter: http://www.informatik.uni-stutgart.de/studieninteressierte/studiengaenge/softwaretechnik.html, Zugriff am 24.01.2014.

Weinert, F. E. (2001): Vergleichende Leistungsmessung in Schulen: Eine umstrittene Selbstverständlichkeit. In: Weinert, F. E. (Hrsg.): Leistungsmessungen in Schulen. Weinheim: Beltz, 17–31.

Lernzentrierte Lehre: Retrieval-Based Learning in der Softwaretechnik

Dominikus Herzberg, Technische Hochschule Mittelhessen
Kerstin Raudonat, Hochschule Heilbronn

dominikus.herzberg@mni.thm.de, kerstin.raudonat@hs-heilbronn.de

Zusammenfassung

Wir Lehrenden lassen die Studierenden mit dem Lernen oft allein, wir halten es für ihre originäre, eigene Angelegenheit – und messen anschließend mit der Strenge einer Prüfung den Erfolg ihrer Lernbemühungen. Die Idee der lernzentrierten Lehre stellt das Lernen in den Mittelpunkt des Interesses. Lernzentrierte Lehre bereitet systematisch auf die Prüfungssituation vor und gibt den Studierenden Lernwerkzeuge an die Hand. Die Lernforschung zeigt, dass Studierende wenig effektive Lernformen nutzen. Dabei ist das Retrieval-Based Learning (RBL) sehr wirksam und leicht umzusetzen. Der Beitrag stellt die Umsetzung von RBL in einer Vorlesung zur Softwaretechnik vor. Die begleitende Untersuchung macht Hoffnung, dass es sich lohnt über eine weitreichende Vision der lernzentrierten Lehre nachzudenken – und weitere Experimente zu machen.

Einleitung: Lernzentrierte Lehre

Die Lernforschung belegt es: Studierende lernen oftmals falsch. Weit verbreitet ist das wiederholte Lesen von Aufzeichnungen und Manuskripten, so lange, bis der Stoff vermeintlich „sitzt" – unglücklicherweise geht damit eine Kompetenzillusion einher. Nachweislich effektiver ist das sogenannte Retrieval-Based Learning (RBL), das prüfungsorientiert aktives Erinnern einfordert; Lernkartensysteme sind z.B. eine bekannte Form des RBL. Es nützt allerdings wenig, den Studierenden ein Lernkartensystem vorzusetzen. Das Lernverhalten wird sich nur dann am RBL orientieren, wenn die Lehre entsprechend darauf ausgerichtet ist. Das folgende Kapitel arbeitet die wissenschaftlichen Grundlagen dazu auf.

Wenn das Retrieval-Based Learning so wirksam ist, dann ist es ein interessanter Versuch, sich mit lernzentrierter Lehre auseinander zu setzen und dazu praktische Erfahrungen zu sammeln. Ein Experiment an der Technischen Hochschule Mittelhessen lotete die Integration von Lernkarten mit Hilfe eines digitalen Lernkartensystems aus. Von diesem Experiment wird in diesem Beitrag ausführlich die Rede sein. Die Studie wurde im Rahmen der Softwaretechnik-Veranstaltung für Informatiker durchgeführt.

Die Softwaretechnik (SWT) ist für ein solches Experiment ein prädestinierter Kandidat. Meist ist hier schon eine Dichotomie gegeben: In einem Teil der Veranstaltung werden praxisnahe und anwendungsorientierte Übungen durchgeführt, oft unter Betreuung von Assistent(inn)en und/oder Tutor(inn)en, so dass Rückmeldungen möglich sind und die Studierenden Korrekturimpulse erfahren. An vielen Hochschulen wird der Praxisteil darüber hinaus in weiterführenden Veranstaltungen ausgebaut zu SWT-Projekten, Lernbühnen (Herzberg & Marsden, 2005a/2005b) etc. An Praxisbezug, verständnis- und anwendungsorientiertem Lehren und Lernen mangelt es in der SWT nicht. Das dokumentieren auch die Bände zur Workshopreihe „Software Engineering im Unterricht der Hochschulen" (SEUH) eindrucksvoll.

Es bleibt in der Regel ein Teil der Softwaretechnik, der nachwievor vorlesungsorientiert ausgerichtet ist und mit einer Klausurprüfung abschließt. In der Softwaretechnik hat sich ein breiter Wissenskanon ausgeprägt, der das Fach und eine dazugehörige Vorlesung sehr wissensintensiv macht. In aller Regel fehlt den Studierenden praktische Erfahrung zur Softwaretechnik z.B. in der Art eines mehrmonatigen Praktikums in der Industrie. Man muss Wissen auf einem weitgehend „unbestelltem Acker" sähen. Aber wie soll das bei den Studierenden verstetigt werden?

Lernzentrierte Lehre heißt, vom Ende her zu denken. Studierende sind elementar am Prüfungsergebnis einer Klausur interessiert. Wenn man versteht, wie Studierende lernen und was sich verbessern ließe, dann kann man die Lehre vom Ende her gedacht entsprechend aufbauen.

Lernforschung und Lernverhalten

Wie lernt ein Student bzw. eine Studentin außerhalb einer Lehrveranstaltung? Was tun Studierende, um den Stoff für eine Prüfung in ihren Köpfen zu verankern und abrufbar zu halten? Die Lernforschung gibt darauf interessante Antworten:

- Studierende wählen in der Mehrzahl Lernformen, die schlechter sind als das sogenannte Retrieval-Based Learning.
- Das Potenzial des Retrieval-Based Learning wird von den Studierenden unterschätzt; vermutlich ist es bei den Lehrenden nicht anders.

Es gibt also gute Gründe, Studierende für das Retrieval-Based Learning zu sensibilisieren und es in die Lehre zu integrieren. So drängt sich eine Frage auf:

- Welche Anreize kann die Lehre setzen, um Studierende für das Retrieval-Based Learning zu gewinnen, und wie kann die Lehre das Lernen für eine Prüfung unterstützen?

Auf diese drei Aspekte wird im Folgenden eingegangen, um das Problem und seine Relevanz zu erläutern.

Wie lernen Studierende?

Cal Newport hat zu seinen Studentenzeiten ein Buch über erfolgreiche Studier-Strategien geschrieben und warnt darin: „Most students incorrectly believe rote review is the only way to study" (Newport 2007, S. 63). Diese Beobachtung wird von der Lernforschung bestätigt (Karpicke et al. 2009): Studierende praktizieren in der überwältigenden Mehrheit das wiederholte Lesen ihrer Mitschriften, des Vorlesungsmanuskripts oder des Lehrbuchs und halten das für eine effektive Lernform. Eine signifikant wirkungsvollere Lernstrategie ist, den Lernstoff aktiv abzuprüfen und zu erinnern – dies wird in der Fachliteratur als Retrieval-Based Learning bezeichnet.

Es ist nicht so, dass Studierende nicht um alternative Strategien wüssten, dennoch bedient sich nur eine absolute Minderheit verschiedener Erinnerungs- und Abfragetechniken. Studierende sind sich nicht darüber im Klaren, dass das Testen oder Abfragen des Lernstoffes gleichzeitig mit einem Lerneffekt einhergeht (*testing effect*). Wenn eingesetzt, dann gilt das Testen bzw. Abfragen eher zur Ermittlung des Lernstands, weniger als Lerntechnik. Karpicke und Kollegen (2009) vermuten, dass die Studierenden Opfer einer Illusion des Kompetenzerwerb sind, der mit dem wiederholten Lesen des Lernmaterials einhergeht: der Text wird zunehmend flüssiger gelesen und bewältigt. Diese Kompetenz-Illusion verleitet Studierende dazu, diese Strategie auch dann anzuwenden, wenn sie alleine für sich lernen.

Newport (2007) empfiehlt in seinem Ratgeberbuch Studierenden eine Methode, die er „Quizz-and-Recall" nennt – es mag wenig überraschen: Es ist eine Retrieval-basierte Technik (a.a.O., S. 59 ff.).

Was ist und was leistet Retrieval-Based Learning?

Der Begriff des Retrieval-Based Learning ist mit „Erinnerungslernen" leidlich übersetzt. Es fehlt der Aspekt, dass die Erinnerung durch eine Aufforderung, meist eine Frage, angestoßen und in Gang gesetzt wird. Die Richtigkeit der gegebenen Antwort oder der Erfüllungsgrad der Aufforderung lässt sich messen und bewerten. Damit grenzt sich Retrieval-Based Learning als „prüforientiertes Erinnerungslernen" deutlich vom reinen „Wiederholungslernen" ab, ohne dass das eine das andere ausschließen würde. Der Lernstoff wird aktiv abgefragt, und es erfolgt eine unmittelbare Rückmeldung.

Beispiele für Techniken, die unter das Retrieval-Based Learning fallen, sind sämtliche Variationen von Frage/Antwort-Spielen wie Quizze, Multiple Choice Tests, Lernkarten usw. Andere Techniken wie das Erstellen von Mindmaps, Concept-Cards oder Texten können zum Retrieval-Based Learning verwendet werden, wenn sie mit einer klaren Aufforderung zur Erinnerung verbunden sind: „Was haben sie eben in der Vorlesung gehört?" oder „Was steht in dem gelesenem Text?". Notizen und sonstige Erinnerungshilfen sind währenddessen tabu.

Wie die aktuelle Lernforschung zeigt, ist Retrieval-Based Learning für das Lernen von besonderer Bedeutung („critical importance"), es ist anderen Lerntechniken in der Regel deutlich überlegen (Karpicke und Roediger 2008). Und es muss mit einem möglichen Vorbehalt aufgeräumt werden: Retrieval-Based Learning eignet sich nicht nur für das Vokabel-Lernen, Faktenwissen oder die Führerscheinprüfung.

In Experimenten wurde nachgewiesen, dass das aktive Erinnern und Rekonstruieren von Wissen ein tieferes Lernverständnis hervorbringt als das Studieren mit der Ausarbeitung von Concept-Maps (Karpicke und Blunt 2011). Der Akt des Retrievals ruft nicht nur Inhalte aus dem Gedächtnis ab, sondern verknüpft Wissen und stellt Sinnzusammenhänge her. Die Wirkmechanismen des Lernens sind sehr gut erforscht (Rösler 2011), und die Ergebnisse zum Retrieval-Based Learning fügen sich gut in die Theorien gut ein (Grimaldi und Karpicke 2012). Es ist an der Zeit, das Retrieval-Based Learning in die Lehre zu integrieren.

Retrieval-Based Learning in der Lehre

Die Empfehlung, Studierende zum Retrieval-Based Learning zu ermutigen, ist naheliegend, greift aber zu kurz (Karpicke et al. 2009). Ohne eine Integration in die Lehre ist den Studierenden wenig Grund gegeben, neue Lerntechniken und Lernansätze im Selbststudium anzuwenden und Lernerfolge zu reflektieren. Lehre, die nicht zum Lernen (ver)führt, verfehlt ihr Ziel: „teaching occurs only when learning takes place“ (Bain 2004). Folgende Maßnahmen geben Beispiele für eine Integration von RBL in die Hochschullehre:

- Jeder Studierende memoriert zu Veranstaltungsbeginn schriftlich den Inhalt der vorangegangenen Vorlesung.
- Während oder am Ende der Vorlesung legt jeder Studierende eine Concept-Map zum Wissensstand an.
- Die Vorlesung stößt zur Reflexion des Verständnisses an, beispielweise durch Kurztests, Verständnisfragen und Quizze.
- Zu komplizierten und schwer zu merkenden Zusammenhängen werden den Studierenden Gedächtnistechniken und Eselsbrücken angeboten
- Die Studierenden identifizieren in Kleingruppenarbeit Leitfragen zur Vorlesungen und formulieren Antworten dazu; die Studierenden erstellen als Teil der Veranstaltung Lernkarten.
- Alternativ werden den Studierenden Lernkarten zur Verfügung gestellt.

Mit solchen Maßnahmen kann der Wert aktivierender Erinnerungstechniken erlebt und eingeübt werden – in der Hoffnung, dass sich das Verhalten beim eigenständigen Lernen daran orientiert. Die Sinnhaftigkeit und Motivation ist freilich nur dann gegeben, wenn die Studierenden um die Prüfungsrelevanz der derart wiederholten und aktiv erinnerten Lerninhalte wissen.

Die Forschung zeigt, dass verschiedene Formate zum Retrieval-Lernen weitgehend gleichwertig sind. Gibt es jedoch unmittelbares Feedback, ob das Erinnerte richtig oder falsch ist, dann steigert das den Lerneffekt (Smith und Karpicke 2013). Lernkartensysteme bringen mehreres zusammen: Retrieval-Learning mit Feedback und Wiederholungen. Wenn Lernkarten in einer Lehrveranstaltung entwickelt und genutzt werden und prüfungsrelevant sind, dann wird ein starker Anreiz zu einem selbstgesteuerten, retrieval-basierten Selbstlernen gesetzt.

Untersuchung zum Einsatz retrieval-basierter Methoden in der Vorlesung „Softwaretechnik“

Es gibt viele Möglichkeiten der Gestaltung lernzentrierter Lehre, die das Retrieval-Based Learning in den Mittelpunkt rückt. Eine Vision, wohin das führen kann, skizziert das vorletzte Kapitel. In den folgenden Kapiteln wird unter den Überschriften „Methodik“, „Evaluation“ und „Diskussion“ ein Experiment ausführlich beschrieben, das den Einsatz von digitalen Lernkarten in der Softwaretechnik vorstellt, auswertet und bewertet. Ziel der Studie ist es, den Ansatz der lernzentrierten Lehre mit einem einfachen Verfahren – einem Lernkartensystem – auszuloten und belastbare Daten zur Auswertung der Erfahrungen zu erheben.

Methodik

Die folgenden Unterkapitel beschreiben den Verlauf der Softwaretechnik-Veranstaltung, den Versuch lernzentrierter Lehre und die letztlich konsequente Ausrichtung der Klausur auf diesen Ansatz. Die Beschreibung schließt neben der Vorlesung die Übung ein, da Studierende mit der Übung für die Klausur relevante Punkte erlangen können.

Der Veranstaltungskontext

Die „Softwaretechnik“ (SWT) ist ein Pflichtfach für Bachelor-Studierende des 3. Semesters im Fachbereich Mathematik, Naturwissenschaft und Informatik (MNI) der Technischen Hochschule Mittelhessen (THM). Die Veranstaltung bedient alle Informatik-Studiengänge des Fachbereichs. Rund 120 Studierende besuchen die Veranstaltung zu Beginn des Sommersemesters 2014.

Die SWT-Veranstaltung teilt sich auf in 2 SWS Vorlesung und 2 SWS praktischer Übung; die Veranstaltung ist 6 ECTS wert. Der Besuch der Übung ist verpflichtend, die Teilnahme an der Vorlesung ist es nicht. Die Übung wird viermal in der Woche angeboten, so dass eine Gruppengröße von etwa 30 Studierenden Freiräume für individuelle Betreuung schafft. Zwei Assistenten unterstützen die Durchführung der Übungen.

Von den 15 Wochen des Sommersemesters sind 14 Wochen mit Übungsaufgaben verplant und gleichmäßig über die Vorlesungszeit verteilt: es gibt sieben Pflichtübungen und – darin eingeschoben – fünf Wahlübungen; zwei der Übungen sind aufgrund des Umfangs auf eine zweiwöchige Bearbeitungszeit angelegt. Die Studierenden erhalten ein, zwei Wochen vor dem Übungstermin die Aufgaben, die zum vorgesehenen Wochentermin abzugeben sind. Wird die Abnahme verweigert, gibt es in aller Regel die Möglichkeit der Nacharbeit und der endgültigen Einreichung in der Folgewo-

che. Die sieben Pflichtaufgaben sind ausnahmslos zu bestehen, um die Berechtigung zur Klausurteilnahme zu erhalten.

Die Übungsaufgaben dienen zur Einübung von Verfahren und Techniken, zur Erarbeitung von Sachverhalten und der Entwicklung von Verständnis. Die Übung ist ein wichtiger und die Vorlesung flankierender Anteil. So geht es z.B. um die Analyse einer einfachen Applikation zur Verwaltung von Musik-Dateien, die Ableitung eines Zustandsdiagramms aus der Anleitung zu einer Stoppuhr, die Zuordnung von Objekt- zu Klassendiagrammen, der Anwendung der Muster Kompositum, Strategie und Visitor etc.

Mit den fünf Wahlaufgaben können Bonuspunkte erzielt werden. Bei Erfolg werden je Wahlaufgabe fünf Bonuspunkte auf die Klausur angerechnet, sofern die Klausur mit mindestens „ausreichend" bestanden ist. Die Klausuren werden gemäß Prüfungsordnung nach einem Punktsystem mit 0 bis 100 Punkten bewertet.[1] Die Bestehensgrenze („ausreichend") liegt bei 50 Punkten. Mit 95 Punkten ist die Note 1.0 erreicht.

Durch die Bonuspunkte haben die Studierenden die Möglichkeit, die Risiken und Unwägbarkeiten, die mit jeder Klausur einhergehen, in Grenzen zu halten. Hat eine Studentin oder ein Student alle Bonuspunkte erlangt (5 x 5 = 25 Punkte) und die Klausur mit 50 Punkten mindestens bestanden, so sind ihr oder ihm 75 Punkte sicher. Dem entspricht die Note 2.4. Die Punkte rechnen sich in Noten mit der Genauigkeit einer Nachkommastelle um.

Die Vorlesung: Anreize setzen zum Retrieval-Based Learning

Die 90-minütige Vorlesung hat ein festes Schema: Zu Beginn werden vier bis sieben Fragen an die Tafel geschrieben oder an die Wand projiziert. Die Studierenden haben etwa zehn Minuten Zeit, die Fragen zu beantworten. Der „Test" simuliert eine Prüfsituation unter den Bedingungen einer Klausur. Es sind nur Papier und Stift erlaubt, es gibt keinen Austausch mit den Sitznachbarn.

Die Fragen greifen den Stoff der letzten Vorlesung auf. Sie dienen der Wiederholung, der aktiven Erinnerung und Reflexion, der Gewöhnung an typische Klausurfragen und der Herstellung von Anschlussfähigkeit an die aktuelle Vorlesung. Die Antworten werden ausführlich besprochen. Meist werden dafür 20 Minuten benötigt. Es bleiben ca. 60 Minuten für die Vorlesung übrig.

Alle besprochenen „Testfragen" gehen in den Fragenpool der Lernkarten ein. Der Test und die Besprechung üben klausurrelevantes Wissen ein und klären die Erwartungshaltung für Antworten. Teils wird beispielhaft erklärt, welche Arten von Antworten aus welchem Grund zu weniger Punkten führen. Oder auch, warum andere Antworten ebenfalls volle Punktzahl erzielen.

Weitere Lernkarten werden aus dem Stoff der aktuellen Vorlesung abgeleitet. Ein Assistent wohnt der Vorlesung bei und macht sich Notizen zu wichtigen oder ausdrücklich betonten „Wissensfragmenten". Aus diesen Notizen entstehen weitere Lernkarten. Pro Vorlesung erweitert sich der Pool an Lernkarten im Schnitt um 20 Fragen und Antworten. Am Ende des Semesters bilden 290 Lernkarten das Wissen aus der Vorlesung ab.

Neben den Lernkarten stehen den Studierenden unmittelbar nach der Vorlesung die Folien als PDF zur Verfügung. Die Verteilung der Folien sowie die Distribution der Lernkarten erfolgt über die hochschulinterne Moodle-Plattform[2].

Die Vorlesung wird Mitte Juni, knapp vor Beginn des letzten Drittels der Vorlesungszeit, evaluiert. Die Evaluation führt eine unabhängige Person des Zentrums für Qualitätsentwicklung (ZQE) der THM in der letzten Viertelstunde der Vorlesung durch. Es kommt der an Hochschulen übliche EvaSys-Fragebogen zum Einsatz.

BrainYoo als Plattform für Lernkarten

Mit ein, zwei Wochen Verzögerung zur jeweiligen Vorlesung erhalten die Studierenden über Moodle ein inkrementelles Update zum aktuellen „Lernkasten".

Die Lernkarten sind mit der Software der Firma BrainYoo erstellt. Dazu ist ein kostenpflichtiger Zugang notwendig. Die Studierenden können die Software zum Lernen kostenfrei nutzen. Es stehen Apps für Android- wie auch iPhone-Geräte zur Verfügung. Es gibt eine Desktop-Variante für den PC und – Onlinefähigkeit vorausgesetzt – eine webbasierte Version der Lernkartensoftware. Gerätevielfalt, taugliche Mobilversionen und ausgereifte Software haben die Wahl auf BrainYoo fallen lassen.

Ein mit BrainYoo erstellter digitaler Lernkasten ist als zip-komprimierte Datei verteilbar und in die Lernumgebung importierbar. Entpackt stellen sich die Lernkarten als leicht lesbare und nachvollziehbare XML-Dateien dar. Die zwischen den `<Question>`- und `<Answer>`-Marken abgelegten Informationen sind im HTML-Format festgehalten. Eingebettete Medien liegen ebenfalls der komprimierten Dateisammlung bei. Die Lernkarten können prinzipiell auch ohne die BrainYoo-Software verarbeitet und aufbereitet werden. Die Offenheit des Formats

[1] http://goo.gl/jVKN2Y (Zugriff am 31. Okt. 2014)

[2] https://moodle.thm.de (Zugriff am 31. Okt. 2014)

ist ein weiterer Fürsprecher für die Wahl von BrainYoo.

Die Klausur

In der letzten Vorlesung bekommen die Studierenden ein klares Versprechen: „Wer die Lernkarten beherrscht, dem ist das Bestehen der Klausur garantiert." Die Lernkarten sind damit ausdrücklich in den Mittelpunkt der Vorbereitungen gerückt.

Bereits erbrachte Leistungen aus den Übungen sind nur in Grenzen Bestandteil der Klausur – damit sind weitreichende Doppelprüfungen von Inhalten ausgeschlossen. So bleiben noch einige Anteile aus der Vorlesung übrig, die nicht durch Lernkarten erfasst und die potentiell klausurrelevant sind. Darauf wird in der letzten Vorlesung hingewiesen.

Zwischen der letzten Vorlesung am Ende des Sommersemesters und dem Klausurtermin liegen zwei Monate. Von den drei Prüfungswochen sind zwei direkt im Anschluss an die Vorlesungszeit; die dritte Prüfungswoche liegt kurz vor dem Beginn der nächsten Vorlesungsperiode. Das soll die Vorbereitungszeiten für die Studierenden entzerren.

Eine Woche vor dem Klausurtermin erfolgt über Moodle eine Benachrichtigung an alle SWT-Studierende, sich zur Vorbereitung besonders mit den Lernkarten auseinander zu setzen.

Die schriftliche Klausur enthält schlussendlich eine Auswahl von 33 der 280 Lernkarten – ein klarer Entschluss, das Lernkarten-Experiment konsequent umzusetzen. Pro „Lernkarte" können maximal drei Punkte erlangt werden. Bei Fehlern werden die Punkte in Einserschritten reduziert. Im ungünstigsten Fall erhält die Studentin oder der Student null Punkte. Minuspunkte sind ebenso ausgeschlossen wie halbe Punkte oder gar Viertelpunkte. Jeder und jede Studierende erhält einen Punkt „geschenkt", um in Summe maximal 100 Punkte erreichbar zu machen.

Zu der 90minütigen Klausur sind keine Hilfsmittel zugelassen. 88 Studierende schreiben die Klausur.

Umfrage

Eine Woche nach der Klausur werden die Noten an das Sekretariat des Fachbereichs gemeldet. Mit der Notenmeldung geht eine E-Mail an all die Studierende, die die Klausur geschrieben haben und in Moodle eingetragen sind. Die E-Mail teilt die Meldung der Note ans Prüfungsamt mit und bittet um die Teilnahme an einer kurzen, freiwilligen Umfrage. Der Link zur Umfrage ist in der Email enthalten. Zwar wird um sofortige Rückmeldung gebeten, dennoch können Studierende auch erst nach Einsicht in das Notenergebnis die Fragen beantworten. Nach zwei Wochen endet die Umfrage.

Die Umfrage und die Erfassung der Antworten ist mit Google Forms realisiert worden. Zu folgenden Fragen wird anonym um Antwort gebeten:

- Wie fair war die SWT-Klausur? (Mit „fair" meine ich: Kam das dran, was Sie erwarten durften und angekündigt war?) – Antwort auf einer Skala von „sehr fair" (1) bis „absolut unfair" (5)
- Wie schwer war die SWT-Klausur? (Mit „schwer" meine ich: Waren die Fragen zu kompliziert, war Ihnen nicht klar, was Sie antworten sollten etc.) – Antwort auf einer Skala von „sehr schwer" (1) bis „gar nicht schwer" (5)
- Waren der Umfang der SWT-Klausur und die zur Verfügung stehende Zeit angemessen – Antwort „Ja" oder „Nein"
- Wieviele Tage haben Sie vor dem Klausurtermin mit dem Lernen begonnen? (Geben Sie die Anzahl der Tage ein. Versuchen Sie, ehrlich zu sein ;-) – Angabe einer positiven Ganzzahl
- Welche Plattformen haben Sie zum Lernen mit den Lernkarten genutzt? – Antworten mit Mehrfachnennungen: „App auf dem Handy", „App auf dem Tablett", „Webbrowser", Freitext für „Sonstiges"
- Wie effizient ist Ihrer Meinung nach das Lernen mit Lernkarten gewesen? (Mit „effizient" meine ich: zu lernendes Wissen ist schnell anzueignen; man weiß sehr gut, wie gut man den Stoff beherrscht.) – Antwort auf einer Skala von „sehr effizient" (1) bis „gar nicht effizient" (5)
- War die Menge der Lernkarten (290) angemessen? – Antwort mit „Ja" oder „Nein"
- Haben Sie außer den Lernkarten zur Klausurvorbereitung … – Mehrfachnennung möglich: „die Folien durchgearbeitet", „die Testfragen zu Beginn einer Vorlesung bearbeitet", „Übungsaufgaben nachgearbeitet", Freitext zu „Sonstiges"
- Womit haben Sie fest in der Klausur gerechnet, was aber dann doch nicht abgefragt wurde? – Antwort als Freitext
- Würden Sie einem MNI-Studenten bzw. einer MNI-Studentin empfehlen, die SWT-Vorlesung zu besuchen? (Auch wenn man die Klausur allein mit dem Lernen der Lernkarten bestehen kann?) – Antwort „Ja" oder „Nein"
- Möchten Sie mir Feedback zur SWT-Klausur hinterlassen? (Haben Sie irgend-

welche Anmerkungen rund um die Klausur?) – Antwort als Freitext.

Alle Fragen sind optional zu beantworten.

Evaluation

Wie viele Studierende haben die Klausurzulassung erhalten und Bonuspunkte gesammelt? Wie viele Studierende haben die Vorlesung besucht? Wie sind die Ergebnisse der Klausur ausgefallen? Und was hat die Umfrage ergeben? Diese Fragen werden in den nachfolgenden Unterkapiteln beantwortet.

Die Übung: Klausurzulassung und Bonuspunkte

Insgesamt haben 96 Studierende mindestens einmal die Übung besucht und eine Pflichtaufgabe eingereicht. 81 Studierende haben mindestens drei Abgaben gemacht. Die Klausurzulassung haben 71 Studierende mit allen sieben Abgaben erfolgreich erhalten.

Wiederholer, die die SWT-Klausur in einem früheren Semester geschrieben und nicht bestanden haben, behalten die vormals erlangte Klausurzulassung. Aus diesem Grund haben so gut wie keine Wiederholer die Pflichtübungen abgelegt.

Es versuchen sich 60 Studierende an den Bonusaufgaben – und daran beteiligen sich auch Wiederholer, um ihre Klausurnote potenziell verbessern zu können. Sie erreichen in Summe 675 Punkte, was im Mittel 11,25 Punkte und damit eine mittlere Verbesserung der Klausurnote um 0,8 ausmacht. Die Standardabweichung von 8,2 deutet auf eine hohe Streuung hin. Es gibt einige Studierende, die nur 5 oder 10 Punkte haben, aber ebenso viele, die 20 oder 25 Punkte erreichen.

In den Freikommentaren zur EvaSys-Evaluation zeigt sich, dass Verbesserungen in der Strukturierung der Übungen und im Umgang mit dem „Massenbetrieb“ angebracht sind.

Die Vorlesung

In Moodle registrieren sich 187 Studierende, zur ersten Vorlesung erscheinen ca. 120 Studierende.

Zählungen werden keine vorgenommen, es herrscht keine Anwesenheitspflicht. Geschätzt kommen zur zweiten und dritten Vorlesung noch um die 100 Studierende, dann pendelt es sich rasch auf 70-80 Studierende ein; das entspricht ungefähr der Zahl der die Übung besuchenden Studierenden.

Im letzten Drittel des Semesters kommen um die 40-50 Studierende. Und das, obwohl es in Woche 14 der 15 Vorlesungswochen noch eine letzte Pflichtübung gibt.

Auf Nachfrage deutet sich an, dass viele Studierende gegen Semesterende eine Überlast durch andere Veranstaltungen empfinden – und die SWT-Klausur ja erst im Oktober sei und man noch genug Zeit zum Lernen daheim hätte. Es scheint, als sei der Besucherrückgang nicht in der Vorlesung an sich begründet.

Die EvaSys-Evaluation weist auf keine Probleme mit der Vorlesung hin. Die Bewertungen sind sehr positiv, das Konzept der Tests zu Beginn der Vorlesung gefällt. Auch die Lernkarten werden positiv aufgenommen. Wenn Kritik geäußert wird, so bezieht sie sich auf die Übung.

Die Klausur: Abgabe und Notenverteilung

Von den 96 gemeldeten Kandidaten treten 88 zur Klausur an; eine Person schreibt die Klausur zeitgleich im Ausland.

Im Mittel werden unter Einrechnung eventuell erlangter Bonuspunkte 66,4 Punkte erreicht bei einer Standardabweichung von 22,9 Punkten. 21 Studierende (23,9%) bestehen die Klausur nicht.

Ohne die durchgefallenen Studierenden ergibt sich ein Mittelwert von 76,5 Punkten (Note 2,3) bei einer Standardabweichung von 13,9 Punkten. Hat ein Studierender die Klausur nicht bestanden, so sind im Mittel 32 Punkte bei einer Standardabweichung von 10,7 Punkten erreicht worden.

Umfrage

Ein besonderes Augenmerk soll auf die Umfrage nach der Klausur gelegt werden. Wenn, dann ist es den Studierenden nach der Klausur möglich, die Veranstaltung in der Gesamtschau zu betrachten und uneingeschränkt Kritik zu üben. Dieser Aspekt wird von den Pflichtevaluationen an den Hochschulen weitgehend ausgeblendet. Eine Rückmeldung nach der Vorlesungszeit ist nicht vorgesehen.

Zur Erinnerung: Mit der Notenmeldung geht eine E-Mail an alle Studierenden, die die Klausur mitgeschrieben haben *und* in Moodle zum Kurs eingetragen sind. Das sind 83 Studierende.

53 Rückmeldungen sind zur Umfrage nach der Klausur eingegangen. Dem entspricht eine Rücklaufquote von 64%. Eine offensichtliche Doppelmeldung ist hierbei schon entfernt. Mit dieser Umfrage konnten mehr Studierende erreicht werden als über die im Rahmen der Vorlesung durchgeführte EvaSys-Evaluation (n=47).

Eine überwiegende Mehrheit von 85% hält die Klausur für „sehr fair“ oder „fair“. Eine Minderheit von 8% stuft die Klausur als „absolut unfair“ bzw. „unfair ein“.

70% halten die Klausur für „gar nicht schwer“ oder „nicht schwer“, weitere 24% halten die Klausur

weder für zu schwer noch für zu leicht. Eine Minderheit von 6% ist mit dem Schweregrad der Klausur nicht einverstanden.

94% der Rückmeldungen halten den Umfang der Klausur und die zur Verfügung stehende Klausurzeit von 90 Minuten für angemessen; der Rest widerspricht. Dieses Ergebnis entspricht dem Eindruck bei der Klausur. Mehr als die Hälfte der Studierenden hat die Klausur nach 45 Minuten bereits abgegeben. Nur wenige Studierende nutzen die gesamte verfügbare Zeit.

Rund zwei Wochen (Mittelwert ist 15 Tage) haben die Studierenden vor der Klausur mit dem Lernen begonnen. Allerdings verrät die Standardabweichung von 12,4 Tagen, dass es ebenso zahlreiche Kurzlerner wie auch langfristig Lernende gibt.

Bei der Lernkartensoftware haben sich 55% der Studierenden der Version für das Smartphone bedient, 26% nutzen ein Tablett zum Lernen, 32% haben mit der Browserversion gearbeitet und 20% benutzen die Desktopversion auf ihrem Rechner. 64% der Studierenden haben dabei entweder nur das Smartphone, das Tablett, den Browser oder die Desktopversion genutzt. 36% der Studierenden arbeiten parallel mit mehreren Varianten und Endgeräten der Lernsoftware.

Als „sehr effizient“ bzw. „effizient“ betrachten 83% der Studierenden das Lernen mit Lernkarten. Weitere 15% sind in der Einschätzung indifferent. Gerade einmal 2% halten Lernkarten für „nicht effizient“. Niemand hält sie für „gar nicht effizient“.

Die Menge an Lernkarten finden 88% der Studierenden als angemessen, 12% sehen das nicht so.

Neben den Lernkarten haben 90% der Studierenden die Klausur mit weiteren Maßnahmen vorbereitet. Von ihnen haben 75% die Folien zur Vorlesung durchgearbeitet, 46% haben die Testfragen zu Beginn einer Vorlesung noch einmal bearbeitet, 54% haben die Übungsaufgaben nachgearbeitet.

29 der 53 Rückmeldungen (55%) machen Angaben, womit Sie in der Klausur gerechnet haben. Einige hielten die Lernkarten-Ankündigung für so deutlich, dass sie das Lernen darauf beschränkt haben. Viele haben Programmieraufgaben und Diagramme verschiedenster Art (Klassen-, Objekt-, Zustands- und Sequenzdiagramme) in der Klausur erwartet.

Den Vorlesungsbesuch würden 94% der Studierenden ihren Kommilitonen empfehlen, selbst wenn man die Klausur allein mit den Lernkarten bestehen kann. 6% der Studierenden würden das nicht tun.

Den Freitext mit Feedback zur Klausur nutzen 25 der Rückmeldungen (47%). 16 Studierende geben ein sehr positives Feedback ab, was die Veranstaltung, die Klausur oder die Lernkarten betrifft. 3 Studierende sind in ihrer Kritik weder besonders positiv noch negativ. Merkliche Kritik äußern 4 Personen, 2 Meinungsbilder fallen differenziert aus.

In den Rückmeldungen wird viermal, teils deutliche Kritik an den Lernkarten geäußert und zwar jedes Mal in ähnlicher Tonlage: jeder könne mit zweiwöchiger Vorbereitung „durch einfaches Auswendiglernen eine gute Note erreichen“, egal ob er oder sie in der Vorlesung gewesen sei oder etwas verstanden habe. Es wäre schön, wenn die „Klausur einen höheren Anspruch gehabt hätte“.

Dem gegenüber stehen andere Kommentare, die den Klausurerfolg auf die Effizienz der Lernens mit Lernkarten zurückführen, die die Vorlesung durch die Lernkarten gut abgebildet sehen, die glauben, viel gelernt zu haben: „Die Lernkarten haben mir beim Lernen sehr geholfen“, „Ich konnte überall lernen.“

Diskussion

Ist lernzentrierte Lehre mit Lernkarten ein geglücktes Experiment für die Softwaretechnik? Das diskutiert dieses Kapitel.

Das Experiment ermutigt, eine lernzentrierte Lehre mit Lernkarten in der Softwaretechnik fortzusetzen und auszubauen. Das Etablieren einer Retrieval-Kultur zu Vorlesungsbeginn sowie die Verteilung von Lernkarten kommen bei den Studierenden gut bis sehr gut an. Das zeigen die EvaSys-Evaluation und die Auswertung der Umfrage nach der Klausur deutlich. Mit Lernkarten und auch der Menge an Lernkarten haben Studierende keine Probleme, sie schätzen diese Lerntechnik als effizient ein.

Fast jeder Teilnehmer bzw. jede Teilnehmerin hat etwas für die Klausur getan – so kann man die relativ hohe Zahl der im Mittel erreichten Punkte interpretieren. Wenn ein Student oder eine Studentin die Klausur bestanden hat, dann in der Regel mit merklichem Abstand von der Bestehensgrenze; die Durchschnittsnote von 2,3 belegt das. Auch spricht die hohe Bestehensquote von rund 75% für die Machbarkeit der Klausur.

Das Ziel, die Lehre vom Ende her und so von der Klausur her zu denken und zu gestalten, darf angesichts von Note und Bestehensquote als Erfolg bewertet werden. Allerdings müsste dieser Effekt in einer Zeitreihenstudie auf seinen Bestand hin untersucht werden. Auch darf man die Messlatte durchaus höher legen und der Vision folgend die Möglichkeiten der lernzentrierten Lehre weiter ausloten: Sind nicht sogar höhere Bestehensquoten und noch bessere Durchschnittsnoten möglich, ohne Kompromisse an die Qualität der Lehre und ihre Inhalte machen zu müssen – Stichwort „Mastery Learning“?

Durchgefallene Studierende erzielen mit einem Drittel der Punkte im Schnitt einen Achtungserfolg, meist ist der Abstand zur Bestehensgrenze jedoch nicht zu leugnen. Anders ausgedrückt: Drei Viertel der Studierenden scheinen mit dem Lernkartenkonzept gut zurechtgekommen zu sein, einem Viertel darf eine falsche Vorbereitung oder ein fehlender Wille zur Auseinandersetzung mit den Lernkarten unterstellt werden.

Bei einer Auswahl von 33 der 290 Lernkarten darf man davon ausgehen, dass die Studierenden, die die Klausur bestanden haben, die Lernkarten umfassend gelernt haben. Bei dieser relativ geringen Auswahl ist es unwahrscheinlich, nur Glück in der Auswahl der Klausurfragen zu haben. Breites Lernen ist als Strategie zur Vorbereitung angemessener als punktuelles Lernen. Insofern dürfte ein Lernziel erreicht sein, die Studierenden mit einem umfassenden Wissen zur Softwaretechnik ausgestattet zu haben.

Aber auch das ist ein wichtiger Punkt: Die Verantwortung für das Lernen geht – bei aller lernzentrierten Lehre – zurück an den Studierenden. Wer sich mit den Lernkarten nicht befasst, hat im wahrsten Sinne des Wortes „schlechte Karten".

Wenn die Möglichkeit zum digitalen Lernen gegeben ist, dann werden Smartphone, Tablett und Browser herangezogen. Die Anwendung auf dem Laptop oder dem heimischen Desktop ist nicht mehr zeitgemäß und verträgt sich nicht mit mobilem Lernen in Bus oder Bahn oder im Urlaub.

Es gibt nur wenige Studierende, die Lernkartensysteme für das „Auswendiglernen" als geeignet, für das Verständnis deutlich ungeeignet halten. Man kann beides vermuten: Äußert sich hier der Unwille, mal etwas wirklich „auswendig" zu lernen? (Was, wie besprochen, deutlich mehr leistet.) Oder ist es der Wunsch, sich über Transferleistungen deutlich besser profilieren zu wollen? An dieser Stelle sind weitere Untersuchungen und Studien angebracht.

Eine Vision: Was digitale Lernkarten ermöglichen können

Die Möglichkeiten der digitalen Lernkarte sind noch lange nicht ausgeschöpft. Insbesondere die Idee einer lernzentrierten Lehre lässt sich mit digitalen Lernkarten auf die Spitze treiben – und sehr gut mit anderen Ansätzen vereinen, wie z.B. dem Inverted Classroom Model (Handke und Sperl 2012). Eine Vision!

Die Ursprünge

Neu ist die Idee der Lernkarte nicht. Ihr historischer Wegbereiter ist die Kartei- oder Registerkarte, die von dem „Vater der modernen Taxonomie", Carl von Linné, im 18. Jahrhundert erfunden wurde, um Wissen effizient mit den Mitteln des Papierzeitalters zu erfassen, zu verschlagworten und zu indizieren. Von der Wissenskarte hin zur Lernkarte ist es nur ein kleiner Schritt. Dokumentiert sind die Lernkarten der englischen Kinderbuchautorin Favell Lee Mortimer aus dem Jahr 1834. Die aus der Lernforschung bekannte Wiederholungsstrategie *Spaced Repetition* und die Lernkarte haben jedoch erst 1973 zum heutigen Lernkasten und dem Leitner-System geführt. Seitdem sind die Vokabelbox und andere thematische Lernkarteien populär.

Neu ist auch nicht die „Elektrifizierung" der Lernkartei. Es gibt Anbieter von Lernkarten-Software, die sowohl im Webbrowser als auch auf mobilen Endgeräten läuft, Geräte gegeneinander synchronisiert und Verknüpfungen mit sozialen Medien anbietet. Kommerziell ausgerichtet sind die Angebote von BrainYoo und StudyBlue; als freie Software ist beispielsweise Anki verbreitet.

Dabei sind die Möglichkeiten digital aufbereiteter Lerninhalte bei weitem nicht ausgeschöpft. Für das vorgeschlagene System sind drei Aspekte neuartig:

- Die Art der digitalen Lernkarte
- Die Beobachtung des Lernens und des Lernprozesses
- Der Einfluss auf die Lehre

Die Art der digitalen Lernkarte

Eine Lernkarte aus Papier hat eine Vorder- und eine Rückseite: vorne steht ein Begriff, eine Aufforderung oder eine Frage, auf der Rückseite die Antwort. In ihrer digitalen Version kann die Lernkarte ähnlich einer Klappkarte um einen Innenteil erweitert werden, mediale Inhalte bereitstellen und generell um dynamische und interaktive Elemente bereichert werden.

- Bei der traditionellen Lernkarte bewertet man die Richtigkeit einer Antwort mit Blick auf die „Rückseite" einer Lernkarte selbst. Interaktive Elemente können die Eingabe einer Antwort verlangen, bei der Mehrfachauswahl (Multiple Choice) die Auswahl der richtigen Antwort einfordern usw. Der Lernvorgang wird durch interaktive Elemente aktivierend und immersiv gestaltet.
- Dynamische Elemente können z.B. Fragen selber generieren, so dass Antworten nicht einfach auswendig gelernt werden. Dynamische Elemente können eine Auswahl an Antworten automatisch erzeugen. Damit ist für Abwechslung und aktives Nachdenken gesorgt.

- Ein „Innenteil“ kann zu einer Lernkarte Erklärungen anbieten und dabei Audios oder Lernvideos einbinden. Der „Innenteil“ erfüllt die Funktion eines multimedialen Lehr- oder Nachschlagewerks, das Lernfragmente in einen Kontext stellt.

All diese Aspekte helfen, das Lernen ansprechend, sinnhaft und effizient zu gestalten. Ganze Lehreinheiten können vollständig in Form derartiger Lernkarten aufbereitet werden. Die Lernkarte bietet den Aufhänger zum Retrieval-Based Learning, ihr Innenteil stellt die Anbindung an ganze Lehreinheiten oder Lektionen her.

Zu einem Lernkartensystem gehört ein Karteikasten mit Fächern, in die gelernte Karten einsortiert und nach einer gewissen Zeit zur Wiedervorlage gebracht werden; dies nennt man „Spaced Repetition“. Jede Form des nachhaltigen Lernens bedarf der Wiederholung, um sich dauerhaft im Gedächtnis zu verankern. Das Leitner-System schlägt *eine* solche Strategie zur Wiedervorlage vor. Ein digitales Lernkartensystem kann beliebige Systeme zur Wiedervorlage von Lernkarten umsetzen. Offenbar ist kein System dem anderen überlegen, doch mögen persönliche Präferenzen oder schlicht ein anstehender Prüfungstermin ein bestimmtes Wiedervorlagesystem favorisieren lassen. Ein digitales Lernkartensystem sollte solche individuellen Wünsche und Konfigurationen berücksichtigen.

Die Beobachtung des Lernens und des Lernprozesses

Wann lernen Studierende: abends, mittags oder morgens? Nutzen Studierende „Totzeiten“, z.B. Pausen zwischen den Vorlesungen zum Lernen? Wann beginnen sie mit dem Lernen für Prüfungen: kurz vor der Prüfung oder noch in der Vorlesungszeit? Wie verteilt sich die Lernintensität und -dauer über einen Tag und über die Tage einer Woche? Welche Lernstrategien erweisen sich mit Blick auf das Prüfungsergebnis als erfolgreich? Ehrlich gesagt wissen Lehrende darüber erstaunlich wenig.

Digitale Lernkarten eröffnen ganz neue Möglichkeiten der Transparenz und versprechen ungekannte Einblicke in den Lernprozess. Es kann erfasst werden, wann ein Student bzw. eine Studentin eine Lernkarte lernt, wie lange er oder sie mit der Lernkarte befasst ist, ob die Lernkarte gekonnt wurde oder nicht, welche Wiederholungsstrategie gewählt wird usw. Wenn Lernkarten mit sozialen Funktionen versehen sind, können Unstimmigkeiten oder Verständnisprobleme erfasst werden. Interessant ist auch, ob und welche Lernkarten sich Studierende zu ihrem Kartensatz hinzufügen.

Es versteht sich von selbst, dass der Datenschutz gewahrt sein muss. Studierende können ihr Lernverhalten protokollieren lassen, müssen es aber nicht – und die Erfassung der Metadaten während des Lernens ist pseudonymisiert, d.h. die Identität des Lernenden bleibt gewahrt.

Ein anderer wichtiger Punkt, der sich mit sozialen Medien koppeln lässt, ist eine zumindest teilweise Gamifizierung des Lernens mit Lernkarten. Das Lernen kann belohnt werden, wenn ein bestimmtes Lernpensum erreicht wird. Der Lernstand im Vergleich mit den Kommiliton(inn)en (aktuellen aber auch mit in der Vergangenheit erfolgreichen Studierenden) mag zur Einschätzung und Korrektur des eigenen Lernverhaltens dienen. Die Gamifizierung mag helfen, die Isolation und die Problematik der Selbstmotivation beim Alleinlernen zu lösen. Davon abgesehen spiegelt ein Lernkartensystem an sich schon Lernerfolge klar zurück und setzt das Prinzip direkten Feedbacks um. Das sind Techniken, derer sich auch die Gamifizierung bedient. Es ist gut möglich, dass Lernkartensysteme keiner ausgeprägten Gamifizierung bedürfen. Dies gilt es in weiterführenden Untersuchung zu analysieren.

Der Einfluss auf die Lehre

Mit ihrem „Innenteil“ erweitert sich die Lernkarte vom Lernmedium hin zum Lehrmedium. Der Inhalt einer Lehrveranstaltung wird portioniert und gleichermaßen eingeklappt in einen „Innenteil“, die relevanten und leitgebenden Fragen werden auf den „Vorderseiten“ eines Lernkartensatzes vermerkt und auf den „Rückseiten“ beantwortet.

Wenn sich die Lehre an diesem Modell der Aufbereitung und Ausrichtung orientiert, dann ist die Lehre hochgradig lernzentriert und damit in letzter Konsequenz auch prüfungszentriert: eine Prüfung ist immer auf das zu Lernende ausgerichtet. Dass es dabei nicht ausschließlich um Lerninhalte von Lernkarten, sondern auch um Anteile aus z.B. Übungen und Hausarbeiten geht, versteht sich von selbst.

Die Lern- und damit Prüfungszentrierung bringt eine unmittelbare Konsequenz mit sich: Richtig gelernt wird vor den Prüfungen. Das ist eine studentische Realität im derzeitigen Prüfungssystem! In der Vorlesungszeit findet die Lehre statt, wozu Vorlesungen, Übungen, Seminare, Praktika, Hausübungen etc. gehören. Auch dann wird gelernt. Doch die intensivste Lernzeit liegt unmittelbar vor den Prüfungen, und meist liegt diese Zeit außerhalb der Lehrzeit oder überschneidet sich nur in Teilen mit dem Ende der Vorlesungszeit. Dann, wenn die Studierenden lernen, fehlt ihnen die Lehrperson.

Der „Innenteil“ einer Lernkarte wird in diesem Moment besonders wichtig. Lehrinhalte können aufgefrischt und nachgeholt werden. Aber das

kompensiert nicht den fehlenden Dialog mit der Lehrperson und die Kommunikation der Studierenden untereinander in der kritischen Phase der Prüfungsvorbereitung. Der Dialog kann ermöglicht werden durch kollaborative Elemente und eine Anbindung an soziale Medien. Zwei Beispiele dazu.

(1) *Welche Lernkarten bereiten den Studierenden Probleme?* – Die Diskussionen auf einer sozialen Plattform (wie z.B. Facebook oder Twitter) um die Art der Fragestellung, die Verständlichkeit und Einprägsamkeit einer Antwort, den inhaltlichen Bezug zum kontextgebenden „Innenteil" einer Lernkarte, ist wichtig im Lernprozess und gibt der Lehrperson Hinweise für Klärungsbedarfe. Updates zu den Lernkarten, verbesserte Fragen, klarere Antworten, überarbeitete Erklärungen während der Lernzeit sind denkbar.

(2) *Welche Lernkarten fügen die Studierenden hinzu?* – Dies ist ein Beispiel für ein kollaboratives Element: Studierende entwickeln ergänzende Lernkarten, um z.B. Inhalte aus Übungen abzudecken, stellen die Karten zur Diskussion und erlauben anderen die Aufnahme in ihren individuellen Kartensatz. Solche Prozesse haben durchaus einen rückkoppelnden und edukativen Aspekt auch für die Lehrperson.

Zum Abschluss

Es sieht so aus, als ließen sich Studierende leichter für Lernkarten in der Softwaretechnik begeistern als gedacht. Die durchgeführte Studie gibt einen Anhaltspunkt, dass es so sein könnte.

Die Vorbehalte liegen vermutlich eher bei den Lehrenden: Man muss Wissen reduzieren und auf den Punkt bringen; man muss ein Gefühl für die Menge an Lernkarten und die Relevanz der Lernkarten entwickeln; man muss bereit sein, Lernkarten als Werkzeug in die Lehre zu integrieren; man muss Lernkartenwissen prüfungsrelevant machen. Vor allem müssen Lehrende mit einem möglichen Vorurteil aufräumen: Es ist nicht so, dass Studierende hirnlos auswendig lernten. Die Lernforschung belegt deutlich, dass beim Retrieval-Based Learning semantische Verknüpfungen und Verständnis aufgebaut werden. Davon abgesehen: ohne Erinnerung, ohne im Besitz des Wissens zu sein, können erst gar keine Verknüpfungen entstehen.

Gerade in der Softwaretechnik können sich Lehrende sehr gut auf ein Lernkarten-Experiment einlassen. Wer Sorge um Verstehens- und Verständnisprozesse hat: der praktische, anwendungsorientierte Übungsanteil in der Softwaretechnik hat einen wichtigen Anteil daran. So ist ein potentieller Experimentalraum geschaffen für Lehr- und Lernexperimente jenseits der traditionellen Vorlesung. Die Vision der digitalen Lernkarte zeigt, dass es noch viel auszuprobieren, zu erkunden, zu erforschen und in seiner Wirksamkeit zu validieren gibt.

Herzlichen Dank an Nils Becker und Artur Klos, die die SWT-Veranstaltung begleitet und das Lernkarten-Experiment möglich gemacht haben.

Literatur

Herzberg, D. & Marsden, N. (2005a): Praxisnahe Förderung von Handlungskompetenz im Software Engineering, in Studienkommission für Hochschuldidaktik an Fachhochschulen in Baden-Württemberg (Hrsg.): Beiträge zum 6. Tag der Lehre, Geschäftsstelle der Studienkommission für Hochschuldidaktik, Karlsruhe, S. 99-103

Herzberg, D. & Marsden, N. (2005b): Das Softwarelabor als Lernbühne: Soziale Kompetenzen im Studiengang Software Engineering praxisnah vermitteln, in Berendt, B., Voss, H.-P., Wildt, J. (Hrsg.) Neues Handbuch Hochschullehre, Ausgabe 04/2005, Berlin: Raabe, S. 1-24

Cal Newport (2007): How to become a Straight A-Student, Broadway Books, S. 63

Jeffrey D. Karpicke, Andrew C. Butler, Henry L. Roediger III (2009): Metacognitive strategies in student learning: Do students practise retrieval when they study on their own?, Memory, 17:4, pp. 471-479

Jeffrey D. Karpicke and Henry L. Roediger III (2008): The Critical Importance of Retrieval for Learning, *Science* 319, 966, DOI: 10.1126/science.1152408

Jeffrey D. Karpicke, Janell R. Blunt (2011): Retrieval Practice Produces More Learning than Elaborative Studying with Concept Mapping, Science 331, pp. 772-775, DOI: 10.1126/science.1199327

Frank Rösler (2011): Psychophysiologie der Kognition – Eine Einführung in die Kognitive Neurowissenschaft, Spektrum Akademischer Verlag

Phillip J. Grimaldi, Jeffrey D. Karpicke (2012): When and why do retrieval attempts enhance subsequent encoding?, Mem Cogn 40:505–513, DOI 10.3758/s13421-011-0174-0

Ken Bain (2004): What the best College Teachers do, Harvard University Press, p. 173

Megan A. Smith & Jeffrey D. Karpicke (2013): Retrieval practice with short-answer, multiple-

choice, and hybrid tests, Memory, DOI: 10.1080/09658211.2013.831454

Handke, J. & Sperl, Alexander (Hrsg.) (2012): Das Inverted Classroom Model, Oldenbourg

British Society for the History of Science. "Carl Linnaeus Invented The Index Card." ScienceDaily. www.sciencedaily.com/releases/2009/06/090616080137.htm (Zugriff am 3. Juli 2014).

Todd Pruzan, Favell Lee Mortimer (2005): The clumsiest people in Europe, or: Mrs. Mortimer's bad-tempered guide to the Victorian world, Bloombury Publishing, p. 5

Sebastian Leitner (2011): So lernt man Lernen – Der Weg zum Erfolg, Nikol, 18. Aufl.

Jeffrey D. Karpicke and Althea Bauernschmidt (2011): Spaced Retrieval: Absolute Spacing Enhances Learning Regardless of Relative Spacing, Journal of Experimental Psychology: Learning, Memory, and Cognition, Vol. 37, No. 5, 1250–1257

Die Essenz des Software Engineering – spielerisch und integriert

Jöran Pieper, Institute for Applied Computer Science, FH Stralsund

Joeran.Pieper@fh-stralsund.de

Zusammenfassung

Softwareprozesse und Software Engineering Methoden gehören zu den Wissensgebieten des Software Engineering (SE), deren anschauliche Vermittlung besonders herausfordernd ist.

Mit der Spezifikation *„Essence – Kernel and Language for Software Engineering Methods“* (Object Management Group, 2014) der *SEMAT Initiative* (SEMAT, 2014) existiert nun ein Ansatz, der u.a. verspricht, alle essentiellen Dimensionen von SE-Aufgaben in einem kompakten universellen und ausführbaren Kernel zusammenzufassen.

Dieser Beitrag beschreibt die Eigenschaften der *Essence*-Spezifikation im Hinblick auf Ihre Eignung, Studierende in die Welt der SE-Methoden einzuführen.

Um ein tiefes Verständnis der *Essence*-Konzepte zu ermöglichen, bedarf es eines geeigneten Ansatzes. Der hier vorgestellte integrierte Ansatz führt Studierende schrittweise in *Essence*-Konzepte ein, lässt sie diese in einer virtuellen Spielumgebung erproben und schließlich erfolgreich praktisch in SE-Aufgaben einsetzen. Dabei unterstützt eine effiziente und vielseitige Lernumgebung bei der aktiven Konstruktion von Wissen. Sie regt die Eigenaktivitäten der Lernenden an, ermöglicht die Betrachtung des Lerngegenstands aus verschiedenen Perspektiven und fördert früh Artikulation und Reflexion im sozialen Austausch. Ziel dieses Ansatzes ist es, Studierende des SE für die Vielfalt der zu berücksichtigenden Dimensionen einer SE-Aufgabe zu sensibilisieren und ihnen darüber hinaus eine wertvolle Orientierungshilfe für die Verwendung von SE-Methoden innerhalb und außerhalb des Curriculums zu erschließen. Dabei wird großer Wert auf die Übertragbarkeit der erlangten Erkenntnisse auf andere Kontexte gelegt.

Einleitung

Probleme / Herausforderungen

Ein idealtypischer Grundsatz konstruktivistischer Didaktik lautet: „Jeder Sinn, den ich selbst für mich einsehe, jede Regel, die ich aus Einsicht selbst aufgestellt habe, treibt mich mehr an, überzeugt mich stärker und motiviert mich höher, als von außen gesetzter Sinn, den ich nicht oder kaum durchschaue…“ (Reich, 2008: S. 95).

Die Notwendigkeit von definierten Prozessen und Methoden in der Softwareentwicklung erschließt sich Studierenden erfahrungsgemäß nicht zwingend intuitiv. Häufig besteht hier die Auffassung, dass (überschaubar komplexe) eigene Projekte in der Vergangenheit mehr oder weniger gut auch ohne "einschränkende Vorschriften" zu bewältigen waren. In Vorlesungen wird eine tiefergehende Diskussion, Analyse bzw. der Vergleich von SE-Methoden durch einen meist (noch) relativ engen Erfahrungshorizont erschwert und begrenzt. In Kursprojekten, welche Lehrveranstaltungen begleiten oder abschließen, können häufig folgende Beobachtungen angestellt werden:

- Es fällt Studierenden nicht leicht, sich in einer gegebenen SE-Methode zu orientieren.
- Es fällt Studierenden schwer, die Frage(n) WER, WIE, WANN und insbesondere WARUM bestimmten Aktivitäten nachgehen sollte, zu beantworten.
- Eine an sich sinnvolle und gewollte Arbeitsteilung innerhalb der Projektteams lenkt den Fokus der Beteiligten auf die Er-

stellung von geforderten Artefakten und führt schnell zur Spezialisierung Einzelner im Team. Es entsteht eine Fixierung auf fachliche und/oder technologische Details. Dabei geht mehrheitlich der Blick für "das große Ganze" der SE-Aufgabe verloren.

- Die gewonnen Erkenntnisse lassen sich häufig nur schwer auf andere Kontexte und kommende Herausforderungen übertragen.

Die Gesellschaft für Informatik e.V. empfiehlt, „… nicht nur gegenwartsnahe Inhalte zu vermitteln sondern auch theoretisch untermauerte Konzepte und Methoden, die über aktuelle Trends hinweg Bestand haben. … Dabei wird … ‚strukturelles' … Denken verlangt. … ‚Strukturelles' Denken … ist mehr-dimensional, es erfordert die gleichzeitige Erfassung mehrerer Entitäten mit ihren strukturellen und Verhaltens-Beziehungen." (Gesellschaft für, Gesellschaft für Informatik e.V. (GI), 2005: S. 8)

Fragestellungen

Mit *Essence* liegt nun ein Standard (Object Management Group, 2014) vor, welcher verspricht, die essentiellen Dimensionen einer jeden SE-Aufgabe (*endeavour*), die notwendigen Aktivitäten und Kompetenzen in einem kompakten, universellen, erweiterbaren und ausführbaren Kernel zusammenzufassen (Object Management Group, 2014). Daraus erwachsen folgende Fragestellungen:

- ***Fragestellung 1:*** Sind *Essence* und der *Essence Kernel* geeignet, Studierende theoretisch und praktisch in SE-Methoden einzuführen?
- ***Fragestellung 2:*** Wie unterstützen *Essence* und der *Essence Kernel* aufgrund ihrer Charakteristik Lehrende in didaktischer und organisatorischer Hinsicht?

Die Förderung eines tiefen Verständnisses von Inhalten und deren Aufnahme in den eigenen Wertekanon sollte Ziel der SE-Ausbildung sein (Ludewig, 2009). Um sich den tiefen Sinn eines Themas zu erschließen, bedarf es einer aktiven kognitiven Auseinandersetzung mit diesem. Daraus folgt:

- ***Fragestellung 3:*** Wie kann eine Lernumgebung gestaltet werden, welche den Einstieg in *Essence* erleichtert, auf den praktischen Einsatz vorbereitet und ein möglichst tiefes Verständnis fördert?

Gliederung

Im Folgenden untersucht dieser Beitrag die o.g. Fragestellungen. Da die *Essence*-Spezifikation vergleichsweise jung ist, werden vorab zentrale Konzepte und Elemente der Spezifikation und des *SEMAT Essence Kernel* in Kürze vorgestellt. Der Beitrag schließt mit einem Fazit, gibt einen kurzen Ausblick und beschreibt die nächsten geplanten Schritte des Sim4SEEd-Projekts.

SEMAT Essence

Die Spezifikation *„Kernel and Language for Software Engineering Methods (Essence)"* (Object Management Group, 2014) ist ein Ergebnis der *SEMAT Initiative* (SEMAT, 2014). Sie liegt aktuell in der Version 1.0 beta 2 vor und definiert einen Kernel sowie eine Sprache für die Erzeugung, Nutzung und Verbesserung vorliegender und zukünftiger SE-Methoden. Die Trennung zwischen Sprache und

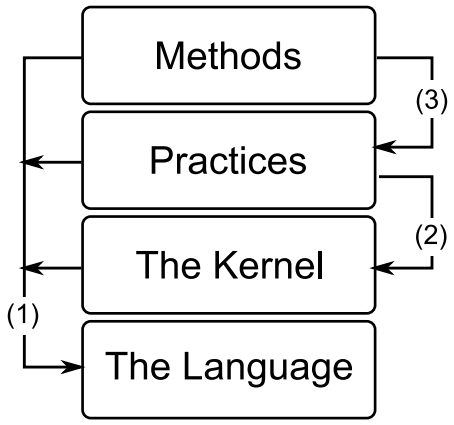


Abb. 1: *SEMAT Essence*: struktureller Aufbau

Kernel unterscheidet *Essence* von vorangehenden Ansätzen wie *SPEM* (Object Management Group, 2008) und *ISO 24744* (International Organization for Standardisation (ISO), 2007). *Essence* zielt nicht vorrangig auf Prozessingenieure sondern auf SE-Praktiker (Object Management Group, 2014).

Abbildung 1 stellt den strukturellen Aufbau des Standards dar. Die Sprache (*Language*) liefert mit einem Metamodell die syntaktische Infrastruktur, mit welcher die grundlegenden Konzepte in einem Kernel abgebildet werden (1). Auf Basis dieser Konzepte werden Praktiken (*Practices*) definiert (2), welche schließlich zu Methoden (*Methods*) zusammengesetzt werden können (3). Der *Essence Kernel* verfolgt nicht das Ziel, jedes denkbare Detail einer jeden verfügbaren SE-Methode abzubilden. Sein Ziel ist es, so klein und universell wie möglich die Essenz dessen abzubilden, was jeder SE-Aufgabe innewohnt. Dazu verwendet der *Essence Kernel* die Konzepte von *Alphas*, *Activity Spaces* und *Competencies*.

Alphas (Abkürzung für *Abstract-Level Progress Health Attribute*) repräsentieren die essentiellen Elemente, deren Fortschritt (*Progress*) und Gesundheit (*Health*) ein jedes Team andauernd im Blick und *Endeavour)* ersichtlich, welche für eine Strukturierung sorgen. Jedes *Alpha* verfügt über eine Menge definierter Zustände, sog. *Alpha States*. Ziel einer jeden SE-Aufgabe ist es, für jedes *Alpha* von einem Anfangszustand über dessen Folgezustände hin zu einem gewünschten Endzustand zu gelangen. Dabei müssen die *Alphas* balanciert weiterentwickelt werden. Eine Checkliste mit *Checkpoints* für jeden *Alpha State* dient als Basis für die Feststellung des aktuellen Zustands eines *Alpha*. Ausgehend vom aktuellen Zustand der *Alphas* ergeben sich deren angestrebten Folgezustände. Die *Alphas* des *Essence Kernel* sind sehr universell gehalten und werden häufig nicht für die detaillierte Beschreibung einer SE-Methode ausreichen. Daher kann der *Essence Kernel* durch Kernelerweiterungen flexibel nach Bedarf ergänzt werden. Die in der Spezifikation enthaltene *Development Extension* bspw. fügt die Sub-Alphas *Requirement Item*, *Software System Ele-*

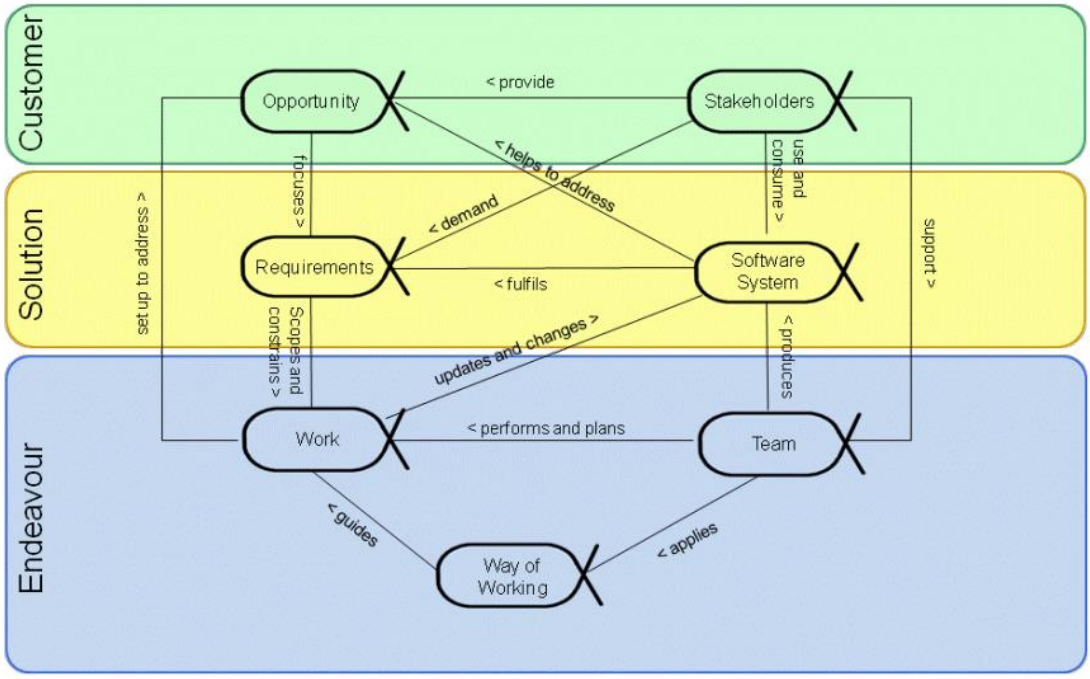


Abb. 2: *Essence Kernel*: *Alphas* und deren Beziehungen (Object Management Group, 2014)

behalten sollte. Sie beschreiben die Dinge, mit welchen immer gearbeitet wird, sobald Software entwickelt, gewartet oder unterstützt wird. Abbildung 2 stellt die 7 *Alphas* des Kernel sowie deren Beziehungen untereinander dar. Die *Alphas* des *Essence Kernel* sind: *Stakeholder*, *Opportunity*, *Requirements*, *Software System*, *Team*, *Work* und *Way of Working*. Aus der Abbildung ist auch die Organisation des Kernel in 3 sog. *Areas of Concern* (*Customer*, *Solution*

ment und *Bug* ein, welche die Kernel Alphas *Requirements* und *Software System* weiter strukturieren und genauer beschreiben. Mit ihrer Verwendung kann zur Laufzeit eines Softwareprojekts nicht mehr nur der Fortschritt aller Anforderungen (*Requirements*) in ihrer Gesamtheit, sondern jeder einzelnen Instanz eines Anforderungselements analysiert und gesteuert werden.

Tabelle 1 stellt in Spalte 3 die *Activity Spaces* des *Essence Kernel* dar. Diese dienen als abstrahierte Platzhalter für konkrete Aktivitäten. Jeder *Activity Space* zielt auf die Erreichung eines oder mehrerer eines bestimmten *Alpha State*. Diese Beziehungen sind in Abbildung 3 zusammengefasst.

Die freie Komposition von *Practices* erfolgt in Methoden *(Methods)*. *Essence* fordert dazu auf, auch

Area of Concern	***Alphas***	***Activity Spaces***	***Competencies***
Customer	– *Opportunity* – *Stakeholders*	– *Explore Possibilities* – *Understand Stakeholders Needs* – *Ensure Stakeholder Satisfaction* – *Use the System*	– *Stakeholder Representation*
Solution	– *Requirements* – *Software System*	– *Understand the Requirements* – *Shape the System* – *Implement the System* – *Test the System* – *Deploy the System* – *Operate the System*	– *Analysis* – *Development* – *Testing*
Endeavour	– *Work* – *Team* – *Way of Working*	– *Prepare to do the Work* – *Coordinate Activity* – *Support the Team* – *Track Progress* – *Stop the Work*	– *Leadership* – *Management*

Tabelle 1: *Essence Kernel*: Die *Areas of Concern* und ihre zugeordneten *Alphas*, *Activity Spaces* und *Competencies*

Alpha States. Aktivitäten erfordern spezifische Kompetenzen *(Competencies)* auf einem definierten Kompetenzniveau *(Competency Level)*. Der *Essence Kernel* definiert 6 Kompetenzen mit verschiedenen Niveaus. Diese sind ebenfalls in Tabelle 1 dargestellt.

Da der Kernel unabhängig von jeder SE-Methode definiert ist, fehlen hier designbedingt methodenspezifische konkrete Aktivitäten *(Activities)* und Arbeitsergebnisse *(Work Products)*. Diese werden erst in Praktiken (*Practices*) auf Basis des Kernels definiert. *Practices* stellen wiederholbare Ansätze dar, begrenzte Aspekte innerhalb einer SE-Aufgabe systematisch und verifizierbar mit einem spezifischen Ziel anzugehen. Eine *Practice* könnte bspw. die Verwendung von *User Stories* sein, um Benutzeranforderungen festzuhalten. Die in *Practices* definierten konkreten Aktivitäten sind jeweils einem *Activity Space* des Kernels zugeordnet. *Activity Spaces* dienen somit der Organisation von *Activities*. Arbeitsergebnisse (*Work Products*) können auf verschiedenen Detaillierungsstufen (*Levels of Detail*) definiert werden. Sie beschreiben praktikspezifisch *Alphas* und dienen als Nachweis für die Erreichung während einer laufenden SE-Aufgabe agil auf neue Entwicklungen und Erkenntnisse zu reagieren, und *Practices* bei Bedarf in die verwendete SE-Methode einzuführen oder auszutauschen.

Die Essence-Spezifikation in ihrer Breite und Tiefe vorzustellen, würde den Rahmen dieses Beitrags sprengen. Hier sei auf weitere Literatur verwiesen (Jacobson u. a., 2013; Object Management Group, 2014; Striewe, Goedicke, 2013).

Kritik

Es soll nicht unerwähnt bleiben, dass die SEMAT-Ideen nicht ohne Kritiker waren (Alistair Cockburn, 2007; Fowler, 2010; Ivar Jacobson u. a., 2010). Ein Großteil der Kritik beruht auf frühen Missverständnissen über die Ausrichtung der SEMAT Initiative. Ein relativ häufiges Missverständnis beruht darauf, *Essence* mit einer bestimmten SE-Methode, z.B. Scrum oder RUP, zu vergleichen. "But the comparisons […] should not be Scrum versus Essence, or CMMI versus Essence, or RUP versus Essence. But rather they should be Scrum versus (Scrum + Essence), or CMMI versus (CMMI + Essence), or RUP versus (RUP + Essence). Essence is not in competition with any existing practice or

method. It is agnostic to your chosen approach and can be implemented by your team today without changing what you are currently doing. It is more about helping you do what you are already doing so you can keep doing it even better in the future." (Paul E. McMahon, 2013)

Sind *Essence* und der *Essence Kernel* geeignet, Studierende theoretisch und praktisch in SE-Methoden einzuführen?

Bei der Beantwortung dieser Frage sollen zwei Aspekte besondere Berücksichtigung finden:

- Wie erleichtert *Essence* den Zugang zu SE-Methoden?
- Welche zusätzlichen Kompetenzen können mit *Essence* vermittelt werden?

Der folgende Abschnitt widmet sich der Beantwortung dieser beiden Fragen.

Wie erleichtert *Essence* den Zugang zu SE-Methoden?

Die Verbreitung von SE-Methoden in unterschiedlichsten Formaten verwehrt Einsteigern einen einheitlichen Zugang. "This can get very confusing [...] as the lack of any common ground between the sources can lead to people saying the same thing in different ways and different things in the same way." (Jacobson u. a., 2013: S. 55) *Essence* bietet ein gemeinsames Vokabular und mit dem Kernel einen einheitlichen Bezugspunkt.

Die Beschreibung von Praktiken auf Basis des Kernels, erleichtert deren Einordnung. Ist der Kernel bekannt, so bezieht sich jede *Essence*-Praktik auf etwas bereits Bekanntes, was das Erlernen einer Praktik nachhaltig erleichtert.

"[Essence promotes] learning and training focused on the essentials..." (Jacobson u. a., 2013: S. 48) - mit *Essence* und dem *Essence Kernel* liegt der Fokus zuerst auf dem Universellen und Essentiellen. Auch ohne alle Details der *Essence Language* oder einer speziellen SE-Methode zu kennen, lassen sich die Konzepte des kompakten *Essence Kernel* sehr gut in jeder SE-Aufgabe verwenden. Die Einstiegshürde – auch zur teilweisen Verwendung – liegt damit sehr viel tiefer. *Essence* priorisiert die tägliche Anwendbarkeit vor einer allumfänglichen detaillierten Prozessbeschreibung und wendet sich damit an eine viel breitere Zielgruppe als vorangegangene Ansätze (International Organization for Standardisation (ISO), 2007; Object Management Group, 2008).

Der Kernel ist auch im physischen Sinne „greifbar“ (*tangible*). Über eine Kartenmetapher lassen sich die wichtigsten Aspekte der *Alphas* und deren *Alpha States* als Karten – ganz wie in einem Kartenspiel – betrachten und tatsächlich anfassen. *Essence* definiert hierzu die Anatomie verschiedener Kartentypen. Druckt man sich die Karten des Kernel oder einer SE-Methode aus, lässt sich bei Diskussionen sehr agil damit arbeiten. So werden bspw. *Alpha State*-Karten einfach verschoben, um gegenwärtige und angestrebte Situationen zu visualisieren. Dies verdeutlicht, dass es sich nicht „nur“ um abstrakte Konzepte, sondern um handfeste praktische Unterstützung bei der Bewältigung einer SE-Aufgabe handelt.

„[Essence] focuses on the needs of the software professional and values the ‘use of methods’ over ‘the description of method definitions’ (the normal priority in the past).“ (Jacobson u. a., 2012: S. 10) Der *Essence Kernel* ist ausführbar (*actionable*). Er beschreibt nicht nur, was getan werden sollte, sondern dokumentiert über die erreichten *Alpha States* auch was tatsächlich bereits getan wurde. Die erlernten Konzepte lassen sich praktisch, bspw. in einem Kursprojekt, erleben. Dabei lässt sich die Frage *„Wo stehen wir gerade?“* über die aktuellen *Alpha States* jederzeit schnell beantworten. Das unterstützt Studierende, welche sich i.d.R. nicht exklusiv um ein Kursprojekt kümmern, sondern parallel andere Lehrveranstaltungen besuchen.

Das Konzept der *Alphas* und ihrer Zustände (*Alpha States*) fordert die andauernde Auseinandersetzung mit den essentiellen Dimensionen der SE-Aufgabe. Fortschritt und Gesundheit aller relevanten Dimensionen werden strukturiert und regelmäßig bewertet. In Kursprojekten bietet dies einen hervorragenden Anlass, im Team über den aktuellen Stand zu diskutieren und neue Ziele zu vereinbaren – ohne sich dabei ausschließlich in fachlichen oder technischen Details des Projekts zu verlieren.

Der *Essence Kernel* hilft dabei, Diskussionen im Team zu steuern. Die verfügbaren Checklisten halten dazu an, über Bereiche nachzudenken, welche zu Problemen heranwachsen könnten – auch dann, wenn das Team dies aus eigener Erfahrung noch nicht vorhersehen würde. Die Checklisten helfen, in der Diskussion die richtigen Fragen zu stellen, um zu besseren Entscheidungen zu gelangen.

Die Beziehungen zwischen *Alphas*, *Alpha States*, *Activity Spaces*, *Activities*, *Work Products* und *Competencies* ergänzt um das Konzept der *Patterns* (sichtbar in Abbildung 3) sorgen dafür, dass sich ausgehend von den aktuellen Zuständen der *Alphas* die notwendigen Aktivitäten zur Erreichung der als nächstes angestrebten Zustände ableiten lassen.

Es ist die Aufgabe von SE-Methoden, zu beschreiben, WER WAS WANN in einer SE-Aufgabe tun sollte. Im Hinblick auf ein angestrebtes tiefes Verständnis, ist insbesondere die Frage nach dem WARUM von Bedeutung. *Essence* bietet hier mit dem Bezug der *Activities* und *Work Products* auf *Alpha States* ein methodenübergreifendes einheitliches Konzept.

Damit liefert *Essence* neben einem einheitlichen Vokabular in vielfacher Hinsicht Orientierung und strukturiertes direkt einsetzbares Expertenwissen in Form von Checklisten. Die Greifbarkeit über die Kartenmetapher nimmt dem Lerngegenstand etwas von seiner Abstraktheit. Der Kernel dient als Wegweiser, regt fortlaufend zur Reflexion an, *WARUM* etwas getan wird und fordert die Betrachtung aller essentiellen Dimensionen. Die definierten Kompetenzen für jede Aktivität verdeutlichen, dass es auch außerhalb der *Solution Area of Concern* wichtige Betätigungsfelder gibt.

Welche zusätzlichen Kompetenzen können mit *Essence* vermittelt werden?

Im Berufsleben werden Studierende auf eine Vielzahl von SE-Methoden und Praktiken treffen. Die Kompetenz, diese neuen Praktiken und Methoden auf einer gemeinsamen Basis einzuordnen, Gemeinsamkeiten und Unterschiede zu erkennen, um daraus eigene Schlüsse zu ziehen, wird im Berufsleben von großem Wert sein. Sie erleichtert die Kommunikation zwischen verschiedenen Projektteams und Organisationen, welche unterschiedliche SE-Praktiken und SE-Methoden verwenden.

Die fortwährende Berücksichtigung aller relevanten Dimensionen in Form von *Alphas* und ihren *Alpha States* schärft die Kompetenz, über die aktuell wichtige Detailaufgabe hinauszuschauen und „das große Ganze“ im Blick zu behalten. Checklisten, welche helfen, die richtigen Fragen zu stellen, unterstützen beim Aufbau einer kritischen Denkweise. Es wird zu zielgerichteter, strukturierter und

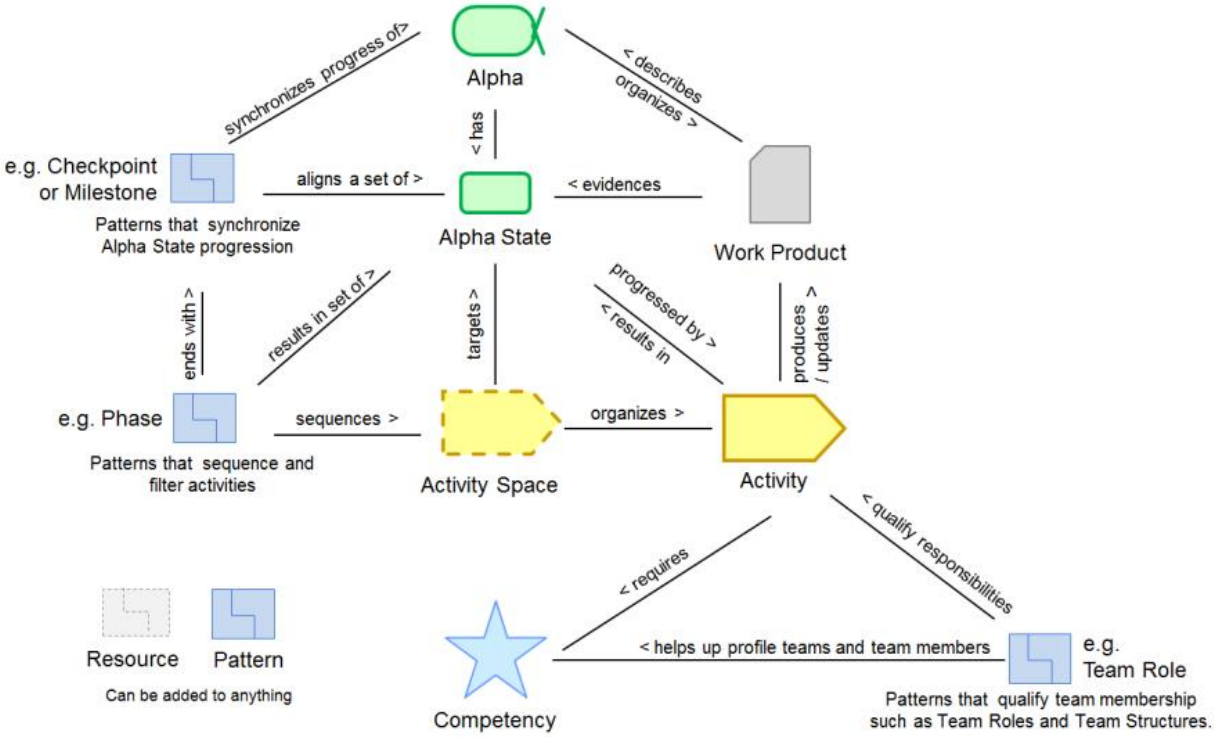


Abb. 3: Essence Language: konzeptioneller Überblick (Object Management Group, 2014)

verifizierbarer Arbeitsweise angehalten – mit einem starken Fokus auf dem Zweck jeder Aktivität.

Die freie Kombination von *Best Practices* zu *Methods* schafft mehr Flexibilität und Freiheit als eher monolithisch anmutende vorangehende Ansätze. Die Einbindung neuer Ideen und Praktiken in bestehende SE-Methoden und in laufenden Softwareprojekten wird dadurch erleichtert. Vermittelt wird hier die Kompetenz, die erlernten Konzepte auf andere Kontexte zu übertragen.

Wie unterstützen *Essence* und der *Essence Kernel* aufgrund ihrer Charakteristik Lehrende in didaktischer und organisatorischer Hinsicht?

Verschiedene SE-Methoden und Softwareprozesse sind unterschiedlich gut für bestimmte Anwendungskontexte geeignet. Ohne sich früh auf einzelne Anwendungskontexte festzulegen, können auf Basis des *Essence Kernel* verschiedene Ansätze vorgestellt werden. *Essence* gestattet es dabei sehr gut, den Detaillierungsgrad einer SE-Methode zu gestalten. So können Anzahl, Breite und Tiefe der vorgestellten SE-Methoden flexibel an den Bedarf von Lehrveranstaltungen angepasst werden.

Veranstaltungsübergreifend bietet *Essence* mit dem Kernel eine gemeinsame Basis und Anknüpfungspunkte. Sind die Grundkonzepte erlernt, so lassen sich verschiedene Praktiken und Methoden didaktisch daraus entwickeln.

Die Arbeit mit *Essence* wird bereits durch digitale Werkzeuge unterstützt, ist jedoch nicht an diese gebunden. So wird eine Vielzahl von Lernarrangements ermöglicht, welche den Lerngegenstand weniger abstrakt erscheinen lassen.

Neu auftauchende SE-Praktiken werden sich auf Basis des Kernels leichter in SE-Methoden integrieren und vermitteln lassen.

Wie kann eine Lernumgebung gestaltet werden, welche den Einstieg in *Essence* erleichtert und auf den praktischen Einsatz vorbereitet?

In der Antwort auf eine Kritik bemerkte einer der SEMAT Initiatoren, „… be prepared to read carefully since it unfortunately takes time to get under the skin of the ideas. “ (Alistair Cockburn, 2007) Wie lässt sich im Rahmen der SE-Ausbildung der Zugang und Einstieg in *Essence* und den *Essence Kernel* erleichtern? Der folgende Abschnitt stellt einen integrierten Ansatz vor, welcher auf die Bereitstellung einer geeigneten Lernumgebung abzielt, um die Eigenaktivitäten der Lernenden anzuregen und die aktive Konstruktion von Wissen zu unterstützen. Die Betrachtung des Lerngegenstands aus verschiedenen Perspektiven, Artikulation und Reflexion im sozialen Austausch tragen grundlegend zum erfolgreichen Lernen bei (Kritzenberger, 2005). Daher stellen diese Aspekte zentrale Ziele des entworfenen Ansatzes dar.

Warum „*spielerisch*“ und „*integriert*“?

„Integriert“ meint im Sinne der Duden-Definition „so beschaffen, dass Unterschiedliches, Verschiedenartiges miteinander verbunden, vereinigt ist“. Im Ansatz, welcher in diesem Beitrag vorgestellt wird, werden aktivierende spielbasierte Konzepte und Simulation dort eingebunden, wo sie aufgrund ihrer Charakteristik einen Beitrag zur Erreichung der Lernziele leisten. Der Begriff „integriert“ passt an dieser Stelle nur bedingt, denn wie der folgende Abschnitt zeigt, sind Spielen und Lernen grundsätzlich nichts Verschiedenartiges. Simulation und spielbasierte Ansätze unterstützen in den Phasen, wo es ohne sie traditionell an Interaktivität mangelt, wo die aktive Konstruktion von Wissen sonst nicht optimal unterstützt wird. Sie tragen zur Erreichung der Lernziele bei, indem Sie eine motivierende, fesselnde Lernumgebung schaffen, die zur aktiven Auseinandersetzung mit einem traditionell eher abstrakten Lerngegenstand einlädt.

In einem Kursprojekt, in welchem die Lernenden aktiv sind, bedarf es aus dieser Sicht keiner weiteren Aktivierung. Hier liefert der vorgestellte Ansatz jedoch Orientierungshilfe und Anlässe zur Artikulation, Diskussion und Reflexion, welche erneut der aktiven Wissenskonstruktion dienen.

Die Phase der theoretischen Einführung und die Phase des praktischen Einsatzes von SE-Methoden werden in diesem Ansatz durch eine Phase verbunden, in welcher die kennengelernten Konzepte in einem Simulationsspiel virtuell erprobt und vertieft werden. Werkzeuge aus dieser Phase des virtuellen Einsatzes finden auch im praktischen Einsatz Verwendung, so dass hier ein weiteres verbindendes Element besteht.

Damit integriert dieser Ansatz Theorie- und Praxisphasen und bindet unterstützende Elemente aus spielbasierten Konzepten ein. Er gibt den Studierenden die Möglichkeit, schrittweise auf Be-

kanntes aufzubauen, das erlangte Verständnis zu erweitern, es zu vertiefen und auf andere Kontexte zu transferieren.

Simulation und digitale Spiele in der Software Engineering Ausbildung

In einem Beitrag der SEUH 2013 (Jöran Pieper, 2013) wurden Vorzüge und Beispiele für den Einsatz von Simulation und Digital Game-Based Learning (DGBL) für den Bereich der Softwareprozesse in der SE-Ausbildung beschrieben. Zusammenfassend lässt sich festhalten, dass Spiel und Nachahmung als natürliche Lernstrategien lebenslang wichtige Akkommodations- und Assimilationsstrategien bleiben (Rieber, 1996). Der fesselnde und motivierende Charakter von Spielen fördert die intrinsische Motivation der Lernenden, indem er sie dazu motiviert, die Verantwortung für den eigenen Lernfortschritt zu übernehmen (Akilli, 2007). Jede Verbindung von Ausbildungsinhalt *(educational content)* und digitalen Spielen wird dabei als *Digital Game-Based Learning (DGBL)* bezeichnet (Prensky, 2007). DGBL fördert den Lernprozess, indem durch die Integration attraktiver Spielelemente – wie Interaktivität, Herausforderungen, kontinuierlichem Feedback, Multimedialität, dem Gefühl der Selbstwirksamkeit, Wettbewerb und Belohnungen – eine motivierende und fesselnde Lernumgebung bereitgestellt wird. „Digitale Spiele können demnach […] ein selbstgeleitetes Lernen durch Exploration ermöglichen und befördern." (Breuer, 2010: S. 12)

Bisherige Forschungsergebnisse zeigten, dass die sorgfältige Planung und Einbettung von Spielaktivitäten in das Curriculum, ausreichende Anleitung, detailliertes Feedback, Diskussion sowie die gründliche Auswertung und Erklärung von Spielresultaten eine essentiell wichtige Rolle für einen Lernerfolg spielen (Wangenheim, Shull, 2009).

Integrierter Ansatz

Der hier vorgestellte integrierte Ansatz gliedert sich in 3 + X Phasen. Die einzelnen Phasen bauen aufeinander auf. Die Phasen 1 bis 3 dienen dabei der Einführung in *Essence* und den *Essence Kernel* sowie der praktischen Erprobung in virtuellen und realen Projektumgebungen. Die Phasen 4 bis X dienen der weiteren Vertiefung und lassen die Lernenden über die Rolle von *Essence Method*-Konsumenten hinauswachsen.

Phase 1: Den *Essence Kernel* kennenlernen

In dieser Phase machen sich Studierende mit *Essence* und dem *Essence Kernel* vertraut. Ziele, Vorteile und die grundsätzliche Struktur des Kernels werden kennengelernt. Als passende Lernmethoden dienen hierbei Literaturstudium, Fallstudien sowie Lehrvorträge in der klassischen Vorlesung. In dieser Phase soll das Interesse geweckt und zur aktiven Auseinandersetzung mit dem Thema angeregt werden. Zum Literaturstudium sind zahlreiche Quellen, einschließlich der gut lesbaren Spezifikation, verfügbar. Auch Fallstudien zur Verwendung des *Essence Kernel* in verschiedenen Projektphasen sind zu finden (Jacobson u. a., 2013).

Für ein interaktives Kennenlernen des *Essence Kernel*, insbesondere der *Alphas* und ihrer Beziehungen untereinander, wurde im Sim4SEEd-Projekt ein Lernspiel entwickelt, welches sich ohne weitere Installationen im Browser ausführen lässt. Spieler sollen mit Hilfe des Spiels in die Lage versetzt werden, die Elemente des *Essence Kernel* zu benennen, Definitionen korrekt zuzuordnen und Beziehungen zwischen den Elementen herzustellen.

Um die Vorlesung in dieser Phase interaktiv, motivierend und fesselnd zu gestalten, eignet sich das kostenfrei online verfügbare *Kahoot!* (Kahoot! AS, 2014) hervorragend. *Kahoot!* ist ein *"game-based blended-learning and classroom response system"* und integriert dabei spielbasierte und soziale Ansätze. Mit *Kahoot!* lassen sich Quiz, Umfragen und Diskussionen vorbereiten und live in die Präsenzveranstaltung integrieren. Es bedarf dafür keiner besonderen Infrastruktur. Der bei einem Quiz entstehende Wettbewerbscharakter weckt Emotionen und wirkt nach Erfahrungen des Autors überaus motivierend. Studierende erhalten unmittelbar Feedback über ihren individuellen Lernstand. Lehrende erhalten unmittelbar einen Eindruck, wie gut einzelne Sachverhalte in der Kursgruppe bereits verstanden wurden. Falsch beantwortete Fragen bieten Anlässe zur Diskussion und können dazu beitragen, Missverständnisse oder Wissenslücken aufzudecken und zu beseitigen. Um den Einstieg zu erleichtern, stellt das Sim4SEEd-Projekt bereits ein vorbereitetes frei verfügbares *Kahoot!*-Quiz zum *Essence Kernel* bereit.

Zum Ende dieser Phase sollten allen Teilnehmenden die grundlegenden Konzepte und Elemente des *Essence Kernel* bekannt sein. Je nach Fokus wurden evtl. schon einzelne Konzepte vertieft.

Phase 2: *Essence* virtuell praktizieren

In dieser nächsten Phase werden die erlernten Konzepte angewendet, um deren Zusammenspiel zu verdeutlichen und die bereits erlangten Erkenntnisse zu vertiefen. Als Lernmethode wird hier ein interaktives kollaboratives Simulationsspiel genutzt, welches im Rahmen des Sim4SEEd-Projekts entwickelt wird.

Lernziel dieses Simulationsspiels ist es, sich in einem simulierten Softwareprojekt mit Hilfe des *Essence Kernel* und einer zuvor gewählten SE-Methode, welche auf Basis des Kernels definiert wurde, zurechtzufinden. Dabei stoßen die Spieler auf Kernel-Erweiterungen und Praktiken (*Practices*), welche den *Essence Kernel* methodenspezifisch ergänzen. Spieler sollen dabei die vorgegebene SE-Methode – wie in einem Adventure-Spiel – erkunden und die entdeckten Methodenelemente einsetzen. Dabei ist es wichtig, die richtigen Dinge (*Activities*) in der richtigen Reihenfolge zu tun und über alle relevanten Dimensionen (*Alphas, Alpha States*) der Aufgabe nachzudenken. Die Spieler erkennen dabei, dass unterschiedliche Aktivitäten (*Activities*) verschiedene Kompetenzen erfordern.

Die Einschätzung der *Alpha States* wird durch die Rückmeldung der virtuellen Spielfiguren ermöglicht. Dabei orientieren sich deren Rückmeldungen an den *Checkpoints* der *Alpha State*-Checklisten aus der gewählten SE-Praktik.

Durch die Zuordnung von Aufgaben an das virtuelle Projektteam, die aktive Verfolgung der Rückmeldungen sowie Analyse der *Alpha State*-Checklisten kann ein Spieler den Verlauf des virtuellen Projekts steuern. Spielfiguren stellen im Spiel wiederkehrend Fragen zu zugeteilten Aktivitäten, *Alphas* etc. Die richtige Beantwortung der Multiple Choice Fragen wirkt sich dabei positiv auf das Spielergebnis aus.

Um sich in der SE-Methode zu orientieren nutzen die Spieler den *Essence Method & Alpha State Navigator*, welcher im Sim4SEEd-Projekt entwickelt wird. Über diesen ist es möglich, durch die SE-Methode zu browsen und die Dokumentation der einzelnen Methodenelemente abzurufen. Gleichzeitig bietet er die Möglichkeit, die *Alphas* im Verlauf der SE-Aufgabe auf die jeweils eingeschätzten *Alpha States* zu setzen. Dabei unterstützen die hinterlegten Checklisten eines jeden *Alpha State*.

Der *Essence Method & Alpha State Navigator* wird mit dem Ziel entwickelt, sowohl in der simulierten Spielwelt, als auch in einem echten Projekt eingesetzt werden zu können. Er dient damit als weiteres verbindendes Element zwischen dieser und der nächsten Phase 3.

Die Konstruktion des dem Spiel zugrunde liegenden Simulationsmodells erfolgt transparent in mehreren Stufen. In der ersten Stufe wird eine verfügbare *Essence Method* ausgewählt. Alternativ wird auf Basis des *Essence Kernel* mit Kernelerweiterungen, verfügbaren oder selbst kreierten *Practices* werkzeugunterstützt selbst eine *Essence Method* definiert. Im Projekt wird dafür gegenwärtig die *EssWork Practice Workbench* (Ivar Jacobson International, 2014) verwendet. Die Verwendung eines Standards und eines Standardwerkzeugs wird im Projekt gegenüber anderen Ansätzen klar favorisiert, denn ein Lernaufwand an dieser Stelle nützt weit über diese Simulations- und Spielumgebung hinaus.

In der nächsten Stufe wird aus der gewählten SE-Methode ein Simulationsmodell generiert. Das Simulationsmodell folgt dabei der Philosophie des *Essence Kernel*. Es ist ebenso konkret und detailliert, wie die gewählte Methode. Enthält die gewählte Methode bspw. detaillierte *Activities* zur Erreichung der *Alpha States*, so werden diese auch im Simulationsmodell abgebildet. Enthält die gewählte Methode keine *Activities* für einen *Activity Space*, so wird stattdessen der generischere *Activity Space* verwendet. Je nach Fokus und Lernziel kann so gezielt die Komplexität des Modells und damit die Komplexität des Spiels gesteuert werden. Anschließend erfolgen notwendige Quantifizierungen innerhalb des Simulationsmodells. Hier wird bspw. festgelegt, wieviel Aufwand in Personenstunden mit einem bestimmten *Competency Level* in eine Aktivität investiert werden muss, um einen bestimmten *Alpha State* oder ein bestimmtes *Level of Detail* eines *Work Product* zu erreichen. Um aus dem Simulationsmodell ein Spiel zu generieren, bedarf es noch eines Spielszenarios, welches bspw. festlegt, in welchem Umfeld das virtuelle Projekt stattfindet und welches virtuelle Personal dafür zur Verfügung steht.

Bisherige Spielansätze im Bereich der Software Engineering Ausbildung setzten auf reine Einzelspielerumgebungen, welche kaum Unterstützung für spielübergreifende Auswertungen und Diskussion bieten. Das Hinzufügen eines Team-Konzepts zum individuellen Spiel jedes Einzelnen schafft hier Mehrwert. Dabei erkunden und bearbeiten alle

Spieler individuell eine ganze SE-Methode. Gleichzeitig jedoch agiert jeder Spieler auch in einem Team und sammelt für dieses Punkte. Indem die Performance des Teams – die Kumulation der Einzelspielerergebnisse – als primärer Erfolgsfaktor herangezogen wird, werden Zusammenarbeit und Diskussion innerhalb der Teams gefördert. Dashboards, Einzel- und Teamrankings bieten dabei zusätzliche Orientierung. Sie zeigen an, wie gut bisher getroffene Entscheidungen in Relation zu anderen Spielern waren. Diese zusätzliche Orientierung dient als Ausgangspunkt für Interaktionen innerhalb der Teams und als Anlass für den ständigen Versuch, Entscheidungen zu optimieren. Das Simulationsspiel bietet an dieser Stelle eine Qualität, welche in Kursprojekten in dieser komprimierten Form nicht geboten werden kann: Lernende erhalten kontinuierlich Feedback zu einem viel früheren Zeitpunkt, als dies in einem realen (Kurs-)Projekt der Fall wäre. Durch die vorgeschlagene Kombination aus Einzelspiel und Zusammenarbeit im Team, kann parallel auch aus Erfahrungen anderer Spieler gelernt und profitiert werden. Das Spiel bietet damit die Basis für einen *„probe, hypothesize, reprobe, rethink cycle"* (Gee, 2007: S. 87 ff.), der zu einer tieferen Auseinandersetzung mit der simulierten SE-Methode führt.

Zum Ende dieser Phase haben die Teilnehmenden die erlernten Konzepte bereits einmal selbst in einem definierten simulierten Kontext angewendet. Dafür haben sie sich notwendigerweise in einer SE-Methode orientiert. Die Spielenden haben alle essentiellen Dimensionen (*Alphas*) der SE-Aufgabe betrachtet und andauernd deren Fortschritt und Gesundheit beurteilt. Die *Alphas* und deren *Alpha States* halfen bei der Orientierung und gaben Empfehlungen für die jeweils nächsten Schritte. Das Spielumfeld, welches Zusammenarbeit und Wettbewerb förderte, hat dazu beigetragen, dass Spielansätze und -züge im Team artikuliert und idealerweise reflektiert wurden. Spielergebnisse wurden kursweit ausgewertet und diskutiert. Damit wurden ideale Bedingungen für die nächste Phase geschaffen.

Phase 3: *Essence* in einem echtem Projekt praktizieren

Inhalt der nächsten Phase ist es, *Essence* in einem realen Projekt einzusetzen. Studierende erfahren in einem Kursprojekt einen bemerkenswerten kognitiven Load. Obwohl es sich i.d.R. „nur" um dem Kursumfang angepasste Aufgabenstellungen handelt, kommen in einem solchen Projekt viele Dinge zusammen. Dieser kognitive Load kann in Teilen überwältigend wirken. Es besteht die Gefahr, dass in einen Deadline-getriebenen Arbeitsmodus verfallen wird, der allein darauf fixiert ist, die geforderten Artefakte zum jeweiligen Termin zu liefern, ohne über die Arbeitsweise, die ausgeführten Aktivitäten und deren Motivation zu reflektieren.

Mit dem *Essence Kernel*, welcher in den zwei vorangegangenen Phasen kennengelernt wurde, steht dem Projektteam nun ein Werkzeug zur Verfügung, welches Orientierung bietet, was dabei unterstützt, im Entwicklungsprozess allen wichtigen Dimensionen die notwendige Aufmerksamkeit zukommen zu lassen und die richtigen Fragen zu stellen. Wird die gleiche SE-Methode wie in der Phase 2 verwendet, können gleiche Aktivitäten wiedererkannt und noch einmal aus einer anderen Perspektive erlebt werden. Dabei kann der bereits in der vorangegangenen Phase kennengelernte *Essence Method & Alpha State Navigator* eine bekannte Umgebung und Orientierung bieten.

Assessment Poker fördert eine reflektierte Arbeitsweise *aller* Teammitglieder. Dabei wird von allen Teammitgliedern unabhängig voneinander eine Einschätzung des gegenwärtigen Projektzustands vorgenommen. Jedes Teammitglied denkt dabei über alle relevanten Dimensionen (*Alphas*) nach und schätzt deren aktuellen Zustand (*Alpha State*) ein. Der Vergleich der individuellen Einschätzungen offenbart schnell unterschiedliche Sichtweisen und ist wertvoller Anlass zum Gedankenaustausch. Wird der Projektfortschritt auch von den Lehrenden zu verschiedenen Zeitpunkten (bspw. Meilensteinen) bewertet, so erfolgt hier ein zusätzlicher Abgleich und Feedback. Die Homogenität der individuellen Einschätzungen kann ein Indiz für die mehr oder weniger erfolgreiche Zusammenarbeit im Team sein und erneut als Anlass für reflektierende Diskussionen dienen.

Der *Essence Kernel* wurde international bereits in Lehrveranstaltungen eingesetzt. Es wurde eingeschätzt, dass sich sein Einsatz positiv auf den Lernerfolg ausgewirkt hat: „By matching the project results against the kernel alphas, the students could easily identify the good and bad sides of their development methods. [...] By following all the kernel alphas, the students could learn the total scope of the software-engineering endeavor and

thereby see what would be required of them in their future as professionals." (Jacobson u. a., 2012)

Zum Ende dieser dritten Phase haben die Studierenden *Essence* in einem realen Projektumfeld eingesetzt. Anders als im virtuellen Umfeld der Phase 2 wurden hier die in der SE-Methode enthaltenen Praktiken tatsächlich angewendet und ausgeführt. Dabei galt es, auch soziale und teamdynamische Aspekte zu bewältigen. Die beschriebenen Maßnahmen sorgten dafür, dass alle Teammitglieder regelmäßig über den aktuellen Projektstatus nachgedacht haben und regelmäßig im Team über den Fortschritt und die Gesundheit der relevanten Dimensionen reflektiert wurde. Somit kann davon ausgegangen werden, dass an dieser Stelle alle Teilnehmenden grundsätzlich in der Lage sind, eine SE-Methode auf Basis des *Essence Kernel* praktisch anzuwenden.

Phase 4 bis X: *Essence* weiter vertiefen

Je nach Ausrichtung des Curriculums bieten sich weitere Aktivitäten im *Essence*-Umfeld an. Eine naheliegende Vertiefung wäre, *Essence* erneut mit einer anderen SE-Methode virtuell zu praktizieren. Im Anschluss können die Ergebnisse und Erlebnisse der Teilnehmer mit denen aus der Phase 2 und Phase 3 verglichen werden.

Studierende werkzeugunterstützt eine eigene SE-Methode kreieren zu lassen, kann sie aus ihrer Konsumenten- und Anwenderrolle in eine Produzenten-Rolle schlüpfen lassen. Die Erzeugung eigener *Practices* und/oder die Komposition von verfügbaren *Practices* zu einer *Method* sind geeignet, das Verständnis von *Essence* über die reine Anwendung vorgefertigter SE-Methoden und Praktiken hinaus zu vertiefen. Die in der Simulation und im Kursprojekt gewonnenen Erkenntnisse lassen sich dabei verwenden. In der anschließenden Diskussion der erzeugten Methoden können verschiedene Standpunkte und Denkweisen thematisiert werden.

Fazit und Ausblick

Der *SEMAT Essence Kernel* erscheint aufgrund der beschriebenen Charakteristik bestens geeignet, Studierende in die Welt der SE-Methoden einzuführen. Besser als vorhergehende Standards ermöglicht *Essence* einen Einstieg, der sich flexibel an den Anforderungen des jeweiligen Curriculums anpassen lässt. Der Fokus auf die essentiellen Dimensionen von SE-Aufgaben und die Priorisierung der täglichen Anwendbarkeit für SE-Praktiker als Entwurfsziel der Spezifikation machen *Essence* zu einem wertvollen Werkzeug und Denk-Framework für alle SE-Studierenden. Der universelle Charakter des *Essence*-Standards und des *Essence Kernel* sorgen für eine hohe Übertragbarkeit der erworbenen Kenntnisse auf andere Kontexte innerhalb und außerhalb des SE-Curriculums.

Der hier vorgestellte integrierte Ansatz führt schrittweise in die Konzepte und die Verwendung von *Essence* ein. Er versetzt Studierende in die Lage, schrittweise die enthaltenen Konzepte kennenzulernen, dabei auf Bekanntes aufzubauen und SE-Methoden erfolgreich praktisch in SE-Aufgaben einzusetzen. Um die aktive Konstruktion von Wissen zu unterstützen, wird dabei eine effiziente und vielseitige Lernumgebung bereitgestellt, welche die Eigenaktivitäten der Lernenden anregt, die Betrachtung des Lerngegenstands aus verschiedenen Perspektiven ermöglicht und Artikulation und Reflexion im sozialen Austausch fördert. Simulation und Digital Game-Based Learning (DGBL) sind dabei wichtige Bausteine.

Mit *Essence* und dem *Essence Kernel* kann über die Grenzen einzelner Lehrveranstaltungen hinaus eine gemeinsame Basis verwendet werden. Interessant wäre es zu untersuchen, ob sich auch ein ganzes Informatik-Curriculum mit Hilfe des *Essence Kernel* strukturieren ließe. Auch Nicht-SE-Kernfächer zur Stärkung von Soft Skills, wirtschaftlichen und organisatorischen Kompetenzen etc. sollten sich bspw. über die *Areas of Concern Customer* sowie *Endeavour* abbilden lassen.

Nächste Schritte

Sim4SEEd (www.sim4seed.org) ist ein laufendes Forschungsprojekt. Der derzeitige Fokus der Projektaktivitäten liegt neben der Implementierung eines *Essence*-Simulationsmodells und der Komplettierung eines Simulationsspiels auf dem Einsatz und der Evaluierung des hier vorgestellten Ansatzes. Dazu ist der Einsatz dieses Konzepts mit anschließender empirischer Auswertung im Sommersemester 2015 geplant. Lehrende, die ebenfalls an einem Einsatz interessiert sind, sind eingeladen, mit Unterstützung des Autors den Einsatz in eigenen Lehrveranstaltungen anzugehen.

Literatur

Akilli, Göknur (2007): „Games and Simulations: A New Approach in Education?“. In: *Games and Simulations in Online Learning: Research and Development Frameworks*. Information Science Pub., S. 1–20.

Alistair Cockburn (2007): „A Detailed Critique of the SEMAT Initiative“. *Alistair.Cockburn.us*. Abgerufen am 14.10.2014 von http://alistair.cockburn.us/A+Detailed+Critique+of+the+SEMAT+Initiative.

Breuer, Johannes (2010): „Spielend lernen? Eine Bestandsaufnahme zum (Digital) Game-Based Learning“. Landesanstalt für Medien Nordrhein-Westfalen (LfM).

Fowler, Martin (2010): „Semat“. *martinfowler.com*. Abgerufen am 14.10.2014 von http://martinfowler.com/bliki/Semat.html.

Gee, James Paul (2007): *What Video Games Have to Teach Us About Learning and Literacy*. Palgrave Macmillan.

Gesellschaft für; Gesellschaft für Informatik e.V. (GI) (2005): „Empfehlungen der Gesellschaft für Informatik e.V. (GI) für Bachelor und Masterprogramme im Studienfach Informatik“.

International Organization for Standardisation (ISO) (2007): „ISO/IEC 24744:2007 - Software Engineering -- Metamodel for Development Methodologies“.

Ivar Jacobson; Bertrand Meyer; Richard Soley (2010): „Some critiques of the Semat initiative | SEMAT blog“. *SEMAT blog*. Abgerufen am 14.10.2014 von http://sematblog.wordpress.com/2010/04/24/some-critiques-of-the-semat-initiative/.

Ivar Jacobson International (2014): „Developing and Customizing Practices | EssWork Practice Workbench“. Abgerufen am 25.06.2014 von http://www.ivarjacobson.com/EssWork_Practice_Workbench/.

Jacobson, Ivar; Ng, Pan-Wei; McMahon, Paul; u. a. (2012): „The Essence of Software Engineering: The SEMAT Kernel“. In: *Queue*. 10 (10), S. 40:40–40:51, DOI: 10.1145/2381996.2389616.

Jacobson, Ivar; Ng, Pan-Wei; McMahon, Paul E.; u. a. (2013): *The Essence of Software Engineering: Applying the SEMAT Kernel*. Upper Saddle River, NJ: Addison-Wesley Professional.

Jöran Pieper (2013): „Alles nur Spielerei? Neue Ansätze für digitales spielbasiertes Lernen von Softwareprozessen“. In: Andreas Spillner; Horst Lichter (Hrsg.) *Tagungsband des 13. Workshops „Software Engineering im Unterricht der Hochschulen“ 2013*. Aachen: CEUR Workshop Proceedings, S. 131–139.

Kahoot! AS (2014): „Kahoot! | Game-based blended learning & classroom response system“. Abgerufen am 30.10.2014 von https://getkahoot.com/.

Kritzenberger, Huberta (2005): *Multimediale und interaktive Lernräume*. München : Oldenbourg.

Ludewig, Jochen (2009): „Erfahrungen bei der Lehre des Software Engineering“. In: Jaeger, Ulrike; Schneider, Kurt (Hrsg.) *Tagungsband des 11. Workshops „Software Engineering im Unterricht der Hochschulen“ 2009*. Heidelberg: dpunkt.verlag, S. 75–86.

Object Management Group (2014): „Kernel and Language for Software Engineering Methods (Essence) Version 1.0“.

Object Management Group (2008): „OMG - Software & Systems Process Engineering Meta-Model (SPEM), v2.0“. Abgerufen am 26.10.2011 von http://www.omg.org/cgi-bin/doc?formal/2008-04-01.

Paul E. McMahon (2013): „Essence: Why do we need it?“. *SEMAT blog*.

Prensky, Marc (2007): *Digital game-based learning*. Paragon House.

Reich, Kersten (2008): *Konstruktivistische Didaktik: Lehr- und Studienbuch mit Methodenpool*. Beltz.

Rieber, Lloyd P. (1996): „Seriously considering play: Designing interactive learning environments based on the blending of microworlds, simulations, and games“. In: *Educational Technology Research and Development*. 44 (2), S. 43–58, DOI: 10.1007/BF02300540.

SEMAT (2014): „Software Engineering Method and Theory“. Abgerufen am 21.10.2014 von http://semat.org/.

Striewe, Michael; Goedicke, Michael (2013): „Modellierung und Enactment mit ESSENCE.“. In: *Proceedings of Workshop „MVV - Modellierung von Vorgehensmodellen - Paradigmen, Sprachen, Tools“, Software Engineering 2013*. Aachen, Germany, S. 405–414.

Wangenheim, Christiane Gresse von; Shull, Forrest (2009): „To Game or Not to Game?“. *IEEE Software*. 26 (2), S. 92–94.

Projektmanagement spielend lernen

Alexander Nassal, Institut für Programmiermethodik und Compilerbau, Universität Ulm
alexander.nassal@uni-ulm.de

Zusammenfassung

Um den Studierenden schon früh ein Gefühl für das Projektmanagement und den damit verbundenen Aufgaben und Probleme zu vermitteln, haben wir ein Spiel entwickelt, das auf dem Projektmanagementwerkzeug MS Project basiert. Unter Verwendung einer umfangreichen Simulation können damit die in MS Project erstellten Projektpläne simuliert werden. Die Studierenden können dadurch sofort erkennen, wie gut ihre Planung ist und wo Probleme auftreten. Ziel dabei ist es, den Studierenden eine Möglichkeit anzubieten, spielerisch und ohne Risiko verschiedene Projektmanagementstrategien unter verschiedenen Bedingungen auszuprobieren. Dabei können die Spieler alle in MS Project vorhandenen Funktionen und Werkzeuge nutzen.

In diesem Beitrag beschreiben wir das Spiel in seinem Aufbau und Ablauf. Um unsere Arbeit besser bewerten zu können, haben wir das Spiel mittels einer Befragung evaluiert. Eine kurze Zusammenfassung dieser Evaluation findet sich am Ende dieses Beitrags.

Motivation

Die Lehre im Bereich der Softwaretechnik, insbesondere im Bereich des Projektmanagements, stellt die Lehrenden vor die große Herausforderung, den Studierenden dieses wichtige Thema verständlich und nachvollziehbar zu vermitteln. Neben der Kenntnis der verschiedenen Methoden und ihren Einsatzfeldern ist es vor allem die Erfahrung, die einen guten Softwareingenieur ausmacht. Diese lässt sich jedoch nicht durch Theorie im Rahmen einer Vorlesung vermitteln, sondern setzt persönliche Aktivität der Lernenden voraus.

Vor allem im Bereich des Projektmanagements ist es aufwändig, den Studierenden dafür ein entsprechendes Übungsangebot zu machen. Ziel des Übungsbetriebs sollte es sein, den Studierenden die komplexen Zusammenhänge in einem Projekt zu verdeutlichen und es ihnen zu ermöglichen ein Gefühl für die Probleme und Schwierigkeiten im Gesamten zu entwickeln.

Im Gegensatz zu technischeren Fächern lassen sich für Projektmanagement nur schwer praxisnahe Beispiele und Übungen finden, an denen die Teilnehmer das zuvor erworbene theoretische Wissen selbst ausprobieren können. Ein reales Projekt benötigt neben dem organisatorischen Aufwand vor allem viel Zeit und zusätzliche Arbeitskapazität zur eigentlichen, vom Projektmanagement unabhängigen Bearbeitung der Projektinhalte. Daher sind solche Projekte als Grundlage einer Übung im Bereich des Projektmanagements nur sehr bedingt geeignet.

Ein möglicher Lösungsansatz ist es, Ausschnitte aus einem fiktiven oder realen Projekt zu verwenden und diese Teilprobleme als Übung zu bearbeiten. Da viele Aspekte sich nicht nur auf einzelne Teile sondern auf das ganze Projekt beziehen, lässt sich damit allerdings nicht alles abdecken.

Da Projektmanagement viel mit Mitarbeiterführung und den damit verbundenen weichen Faktoren zu tun hat, ist es oft schwierig konkrete Regeln oder Handlungsvorgaben für einzelne Situationen anzugeben. Die Studierenden müssen lernen, entsprechende Situationen zu erkennen, zu bewerten, zu lösen und dabei das Projekt als Gesamtes zu berücksichtigen. Dabei geht es nicht nur um das zu erstellende Produkt, sondern auch um die beteiligten Mitarbeiter und weitere Aspekte, wie beispielsweise die Marktreputation des Unternehmens.

Als einen ersten Ansatz zur Bewältigung der oben beschriebenen Problematik haben wir das hier vorgestellte Planspiel entwickelt. Dazu haben wir die eigentliche, sonst aufwändige und teure Projektarbeit als Simulation in das Projektmanagementwerkzeugs *Microsoft Project* (Microsoft Corporation, 2013) von Microsoft (im Folgenden als *MS Project* bezeichnet) integriert. Damit ermöglichen wir den Studierenden, das in MS Project geplante Projekt auch tatsächlich durchzuführen und durch geeignetes Feedback des Spiels zu sehen, wie erfolgreich die eigene Planung war. Im Gegensatz zu einem realen Projekt können dabei problemlos und in kurzer Zeit verschiedene Lösungsansätze ausprobiert und verglichen werden.

Durch die Integration des Spiels in eine umfangreiche und weit verbreitete Software stehen den Spielern viele ausgereifte Werkzeuge der modernen Projektplanung zur Verfügung, ohne dass diese speziell für das Spiel entwickelt werden müssen. Den Spielern ist es so beispielsweise möglich, die in MS Project vorhandene Ressourcenplanung zu verwenden, um zu sehen, wie gut die einzelnen Mitarbeiter ausgelastet sind, und um schnell zu erkennen, ob einzelnen Mitarbeitern zu viele Aufgaben aufgetragen wurden. Als positiven Nebeneffekt lernen die Studierenden dabei auch den Umgang mit MS Project.

Dieser Beitrag ist wie folgt strukturiert: Nach einem kurzen Überblick über bestehende Ansätze und verwandte Arbeiten folgt eine Beschreibung der Plattform und des verwendeten Simulationsframeworks. Der Hauptteil beschreibt den Aufbau und Ablauf des Spiels und liefert ein mögliches Beispielszenario, auf dem die anschließende Evaluation basiert. Zum Schluss bilden wir ein kurzes Fazit und diskutieren mögliche Verbesserungen und Erweiterungen, die bislang noch nicht umgesetzt wurden.

Verwandte Arbeiten

Vorhandene Arbeiten zeigen, dass die Idee, Simulationen und Spiele in der Softwaretechniklehre einzusetzen, nicht neu ist. Die meisten Projekte fokussieren dabei einen bestimmten Schwerpunkt. So konzentrieren sich beispielsweise *SimjavaSP* (Shaw u. Dermoudy, 2005) auf Wasserfall- und Spiralmodell und *SimVBSE* (Jain u. Boehm, 2006) auf die Thematik des Value-Based Engineering.

Durch eine Storyline und zusätzliche Spielinhalte kann der Spielspaß entsprechend erhöht werden, wie *The Incredible Manager* (de Oliveira Barros u. a., 2006) zeigt. Es gab in der Vergangenheit auch Versuche, Lernaspekte in bestehende Spiele wie beispielsweise *Second Live* zu integrieren (Ye u. a., 2007).

Andere Projekte verfolgen den Ansatz, eine Plattform zur Verfügung zu stellen, mit der eigene Lerninhalte selbst gestaltet und – meist in Form eines Simulationsmodells und zusätzlichem Inhalt für die geeignete textuelle und graphische Präsentation – zu einem Spiel zusammengesetzt werden können. Eines der ersten Projekte dieser Kategorie war das *SESAM* Projekt (Ludewig u. a., 1992) mit einem Schwerpunkt auf dem Qualitätssicherungsaspekt im Softwareentwicklungsprozess. Das umfangreichste uns bekannte Projekt auf diesem Gebiet ist *SimSE* (Navarro, 2006) der University of California, Irvine, das neben vielen Beispielszenarien für die unterschiedlichsten Vorgehensmodelle auch umfangreiche Editoren zur Erstellung eigener Inhalte bereitstellt.

Alle oben genannten Arbeiten versuchen einen oder mehrere Schwerpunkte des Software Engineering mit Hilfe einer eigens dafür entwickelten Spielumgebung zu vermitteln. Dabei können Schwierigkeitsgrad und benötigte Hilfestellungen gut an die Vorkenntnisse und Bedürfnisse des Spielers angepasst werden. Durch die künstlich geschaffene Spielumgebung wirkt ein solches Spiel aber oft aufgesetzte und realitätsfern. Dieser Effekt kann durch ein einfach gehaltenes Simulationsmodell verstärkt werden, das sich auf den Kern des zu vermittelnden Schwerpunkts beschränkt und keine Randeffekte berücksichtigt.

In der Arbeit *Alles nur Spielerei? Neue Ansätze für digitales spielbasiertes Lernen von Softwareprozessen* (Pieper, 2013) werden die oben genannten Projekte anhand verschiedener Kriterien verglichen. Die Arbeit gibt außerdem einen tieferen Einblick in die Vor- und Nachteile der einzelnen Spiele und zeigt ungenutztes Potential auf.

Einer von Piepers Kritikpunkten ist die oft fehlende Integration von Werkzeugen aus der realen Softwareentwicklung in die Simulationsspiele. Für unser Projekt haben wir den umgekehrten Weg gewählt und die Simulation direkt in das Werkzeug MS Project integriert, um den Aspekt des Projektmanagements, insbesondere der Aufgabenplanung, für die Studenten erfahrbar zu machen.

Plattform und Simulationsframework

Um die Simulation nahtlos in MS Project zu integrieren, wurde das Spiel als AddIn unter Verwendung der Office-API entwickelt. Nach der Installation des AddIns stehen damit alle Funktionen direkt in MS Project zur Verfügung, weitere Werkzeuge werden nicht benötigt. Das AddIn kann über die API auf nahezu alle Funktionen und Daten von MS Project zugreifen und somit die Verbindung zwischen MS Project und dem Simulationsframework herstellen.

Um eine umfassende und realitätsnahe Simulation der Vorgänge in einem Projekt durchführen zu können, haben wir das in (Nassal, 2014) vorgestellte Framework für Planspiele im Bereich des Softwareengineering verwendet. Dieses Framework wurde speziell für den Einsatz in Lernspielen im Bereich der Softwaretechniklehre entwickelt und enthält neben den notwendigen Simulationsmodellen alle weiteren Elemente, die wir für die Simulation in unserem Spiel benötigen.

Das Simulationsmodell des Frameworks geht deutlich über die einfachen Berechnungen hinaus, wie sie in einigen vergleichbaren Spielen zu finden sind. Stattdessen wurden Modelle basierend auf Erkenntnissen der Arbeits- und Lernpsychologie verwendet, und zu einem komplexeren Gesamtmodell kombiniert. So werden beispielsweise die charakterlichen Eigenschaften der Mitarbeiter, ihr Lernverhalten, ihr Gesundheitszustand und ihre Belastbarkeit berücksichtigt. Ein eigenes Modell für die Motivation regelt die Arbeitsleistung und das Verhalten der Mitarbeiter. Der Projektplan wird dabei lediglich als Vorgabe verwendet, das tatsächliche Verhalten der Mitarbeiter kann je nach Situation entsprechend davon abweichen. Das spiegelt die Realität besser wider, als wenn der Projektplan strikt nach Vorgabe abgearbeitet wird.

Durch die Verwendung eines detaillierten Modells wird es für den Spieler notwendig, sich neben der Aufwandsschätzung, Abhängigkeits- und Terminplanung auch Gedanken über den richtigen Einsatz seiner Mitarbeiter zu machen, um ein vorgegebenes Problem erfolgreich lösen zu können.

Das Framework arbeitet auf einer Datenstruktur, die das Vorgehensmodell, die Rahmenbedingungen des

Projekts, die dazugehörigen Mitarbeiter mit ihrem jeweiligen Charakter und weiteren Eigenschaften, sowie das zu entwickelnde Produkt festlegt. Die Datenstruktur wird dabei in Teilen von der Simulation bei deren Ausführung fortgeschrieben. Des Weiteren kann diese Datenstruktur auch während der Simulation verändert werden, um beispielsweise besondere Ereignisse wie den Ausfall eines Mitarbeiters oder veränderte Anforderungen an das Produkt zu realisieren.

Das Vorgehensmodell legt fest, welche Aktivitäten unter welchen Bedingungen durchgeführt werden können. Als Basisaktivitäten stehen *Entwicklungsarbeit*, *Qualitätsarbeit* und *Kommunikation* zur Verfügung. Jeder Aktivitätstyp wirkt sich unterschiedlich auf das zu entwickelnde Produkt und die Mitarbeiter aus. Entwicklungsarbeit bewirkt einen Fortschritt bei den Produktartefakten. Qualitätsarbeit findet und behebt Fehler, die durch Entwicklungsarbeit in den Artefakten entstanden sind und erhöht damit die Qualität der Artefakte. Kommunikationsarbeit hat keinen Einfluss auf das Produkt, sondern dient dazu, Wissen zwischen den einzelnen Mitarbeitern oder Gruppen von Mitarbeitern auszutauschen.

Wir erzeugen die benötigte Datenstruktur aus einem Szenario, das über das AddIn geladen werden kann. So können ohne großen zusätzlichen Aufwand unterschiedlichste Szenarien für unser Spiel erstellt werden, ohne das AddIn anpassen zu müssen.

Let's play

Im Folgenden wird das Spiel in seinem Aufbau und Ablauf beschrieben und die Entwurfsentscheidungen, die dazu geführt haben, erläutert.

Da ein Wettbewerb zwischen den Studierenden oft eine gute Motivation ist, haben wir für das Spiel ein Score entwickelt, der den Projekterfolg misst und vergleichbar macht. Er liefert außerdem ein gutes Feedback für die Studierenden und ermöglicht es ihnen, ihr Vorgehen besser zu bewerten. Der Aufbau dieses Score wird im Anschluss an die Beschreibung des Spiels erläutert.

Am Ende dieses Abschnitts steht ein Beispielszenario, das den Spielaufbau noch einmal verdeutlichen soll und außerdem als Grundlage für die Evaluation des Spiels gedient hat.

Spielaufbau und Ablauf

Zu Beginn eines Spiels muss ein vorgefertigtes Szenario geladen werden. Ein Szenario enthält neben den Simulationsparametern und den Rahmenbedingungen des Projekts das zu verwendende Vorgehensmodell, welches wiederum die möglichen Aktivitäten, Rollen und Artefakttypen vorgibt. Es enthält außerdem das zu entwickelnde Produkt und damit die Menge der Artefakte, die während des Spiels erstellt werden müssen.

Je nach Lernziel kann im Szenario ein Projektteam definiert sein, mit dem der Spieler das Spiel beginnt. Sind keine Mitarbeiter vorgegeben, muss der Spieler sich diese über den Arbeitsmarkt zusammenstellen.

Der Arbeitsmarkt wird ebenfalls über das Szenario konfiguriert. Hier können Parameter wie Durchschnittsgehalt, Durchschnittsfähigkeiten, die Anzahl der verfügbaren Arbeitskräfte und Ähnliches festgelegt werden. Der Arbeitsmarkt kann auch vollständig deaktiviert werden. In diesem Fall ist der Spieler auf den vorgegebenen Mitarbeiterstamm beschränkt und kann diesen nicht verändern.

Eines der Lernziele des Spiels ist es, dem Spieler ein bestimmtes Vorgehensmodell und die damit verbundenen Aktivitäten und Rollen zu vermitteln. Das Vorgehensmodell ist im Szenario definiert und kann vom Spieler im Spiel nachgeschlagen werden. Dafür kann im Szenario neben den Parametern der Aktivitäten und Rollen jeweils ein Beschreibungs- und Erklärungstext hinterlegt werden. Zusätzlich zu den Texten werden auch alle relevanten Abhängigkeiten angezeigt. Dargestellt werden beispielsweise die Artefakttypen, die mittels einer bestimmten Aktivität bearbeitet werden können, die Rollen, die für eine Aktivität zuständig sind und weitere Artefakttypen, die einen potentiellen Einfluss auf die Bearbeitung haben können, falls das im Produkt entsprechend definiert wurde.

Abbildung 1 zeigt die *Entwurfsaktivität* des weiter unten beschriebenen Beispielszenarios. Bearbeitet werden können hier sowohl Artefakte mit dem Typ *Entwurfsdokument*, als auch Artefakte mit dem Typ *UI Spezifikation*. Außerdem ist zu erkennen, dass es sich um eine Entwicklungsaktivität handelt und damit ein Dokument neu erstellt oder weiterentwickelt wird.

Analog zur Beschreibung des Vorgehensmodells kann der Spieler auch eine Beschreibung des Produkts einsehen. Hier werden alle Teilartefakte des Produkts aufgelistet und beschrieben.

Neben dem Beschreibungstext – der erklärt, um was für ein Dokument es sich handelt, für was es verwendet wird und wie es erstellt werden kann – ist auch der Artefakttyp angeben, der den Bezug zum Vorgehensmodell herstellt.

Weiterhin sind die im Szenario definierten Kontextartefakte und etwaige Voraussetzungen für die Bearbeitung aufgelistet. Ein Kontextartefakt ist ein anderes Artefakt, dessen Vollständigkeit und Qualität einen Einfluss auf die Erstellung des neuen Artefakts hat. So hat beispielsweise der Architekturentwurf einen Einfluss auf die Implementierung. Eine Voraussetzung ist eine Bedingung, die vorgibt, welchen Zustand andere Artefakte haben müssen, damit das neue Artefakt bearbeitet werden kann. So können logische und zeitliche Abfolgen in der Entwicklung sichergestellt werden. Beispielsweise kann Quellcode erst dokumentiert werden nachdem er erstellt wurde.

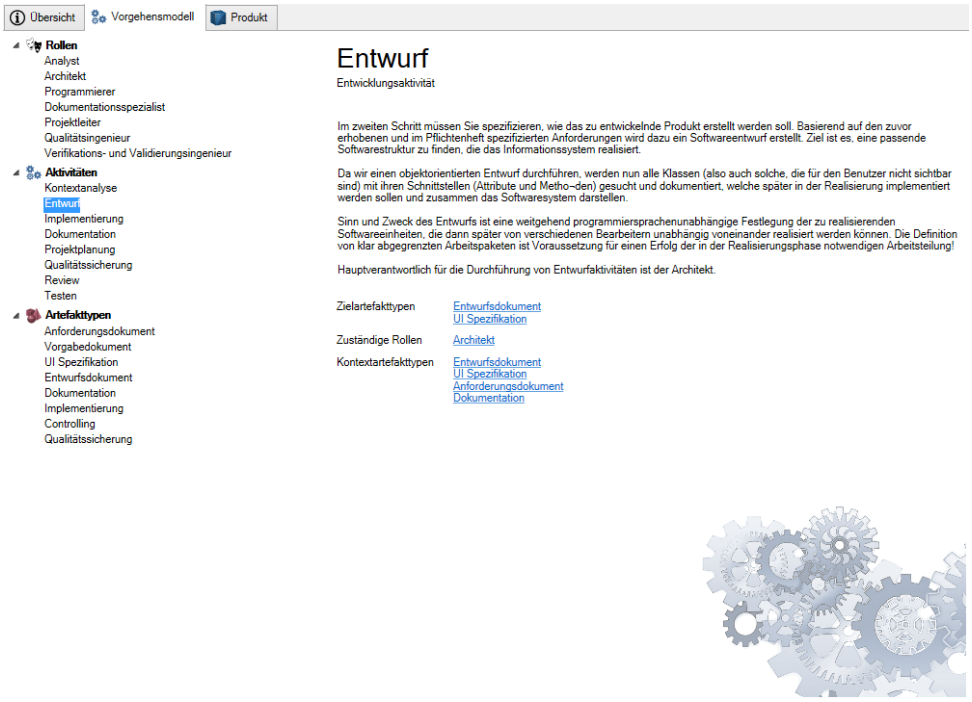


Abbildung 1: Erläuterung zur Entwurfsaktivität in der Szenariobeschreibung

Abbildung 2 zeigt das Produktartefakt *Dialoggestaltung* aus dem unten beschriebenen Beispielszenario. Es liefert dem Spieler alle Informationen, die notwendig sind, um das zu entwickelnde Artefakt zu verstehen und es in den Produkt- und Projektkontext einordnen zu können. Dazu werden neben einem beschreibenden Text auch die relevanten Kontextartefakte des Produktartefakts, sowie die für die Bearbeitung notwendigen Voraussetzungen und relevanten Fachbereiche aufgeführt.

Vorgangsplanung

Der Kern des Spiels ist der Projektplan, anhand dessen die Mitarbeiter entscheiden, wann sie welche Tätigkeit durchführen. Die simulierten Mitarbeiter entscheiden hauptsächlich anhand dieses Plans, welche Basisaktionen sie in der ihnen vorhandenen Zeit durchführen.

Den Projektplan erstellen die Spieler mit den in MS Project vorhandenen Werkzeugen. Das Ergebnis ist die Menge der Vorgänge, welche in MS Project typischerweise als Gantt-Diagramm dargestellt wird (Abbildung 3).

Das AddIn wandelt die Vorgänge aus MS Project in simulationsinterne Aufgaben des Frameworks um. Ein Vorgang in MS Project besitzt folgende für die Simulation wichtige Attribute:

- Einen Start- und Endzeitpunkt
- Eine Tätigkeit die ausgeführt werden soll
- Ein Artefakt auf dem die Tätigkeit ausgeführt werden soll
- Einen oder mehrere Mitarbeiter, die dem Vorgang zugewiesen wurden

Tätigkeit und Artefakt können dabei aus einer vorgegebenen Liste ausgewählt werden, die über das Szenario festgelegt ist. Mitarbeiter werden einem Vorgang, wie in MS Project üblich, als Ressource zugeordnet. Das erlaubt es, die in MS Project vorhandenen Werkzeuge und Ansichten zur Ressourcenplanung zu nutzen, um damit leichter eine optimale Auslastung zu erreichen und eine Überlastung der Mitarbeiter zu verhindern.

Neben der Unterstützung zur Ressourcenplanung können auch alle anderen in MS Project integrierten Werkzeuge zur Vorgangsplanung verwendet werden. Dazu gehören unter anderem die zeitlichen Abhängigkeiten der einzelnen Vorgänge, sowie deren Gruppierung und Hierarchisierung. Auch Hilfsmittel wie Beschriftung, Einfärbung und Kommentierung von Vorgängen stehen zur Verfügung.

Die Steuerung der Simulation ist sehr einfach. Der in MS Project vorhandene Cursor, der das aktuelle Datum in einem Projektplan anzeigt und somit Vergangenheit und Zukunft voneinander trennt, markiert den aktuellen Zeitpunkt der Simulation. Wird ein neues Zeitintervall simuliert, läuft der Cursor bis zu diesem Zeitpunkt vor.

Die Simulation wird über das Simulationsribbon (siehe Abbildung 4) in MS Project gesteuert. Neben der Möglichkeit ein neues Szenario zu laden, bzw. das geladene Szenario auf die Ausgangssituation zurückzusetzen, beinhaltet es verschiedene Möglichkeiten die Simulation bis zu einem gewünschten Zeitpunkt auszuführen. Dabei werden die vom Spieler festgelegten Vorgänge verwendet, um den Fortschritt und

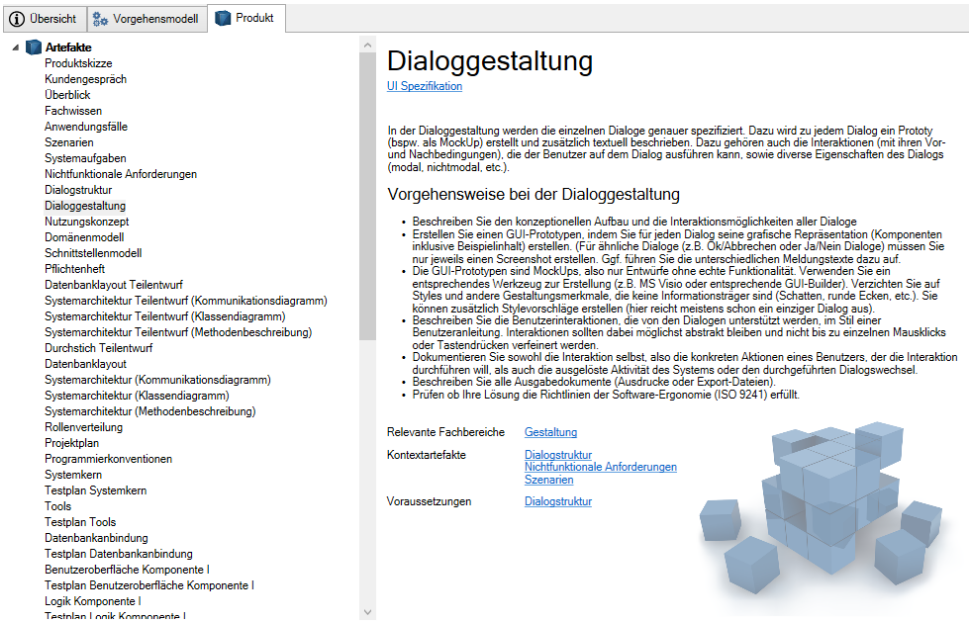


Abbildung 2: Erläuterung zur Dialoggestaltung in der Szenariobeschreibung

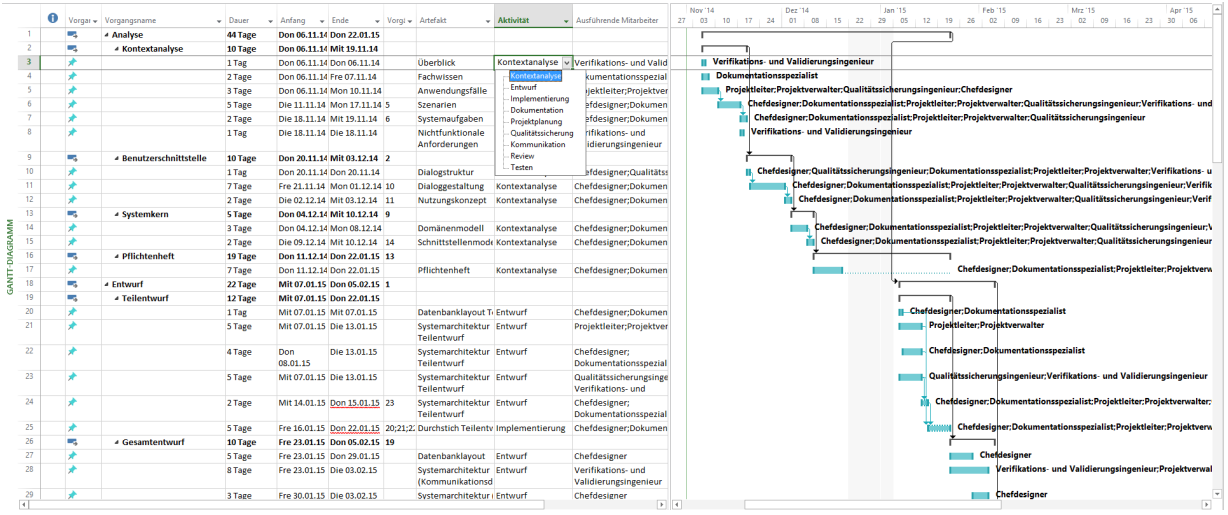


Abbildung 3: Vorgangsplanung in MS Project

die Qualität der Artefakte entsprechend des Simulationsmodells und der Parameter, welche im Szenario festgelegt wurden, zu aktualisieren.

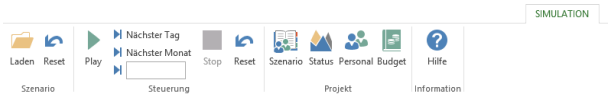


Abbildung 4: Steuerelemente der Simulation

Um dem Spieler zu ermöglichen, aus seinen Fehlern zu lernen, diese zu korrigieren und die Auswirkungen der Korrekturen zu sehen, kann die Simulation auf den Anfangszustand zurückgesetzt werden. Geplante Vorgänge bleiben dabei, im Gegensatz zum Zurücksetzen des Szenarios, erhalten. Merkt der Spieler beispielsweise, dass er seine Mitarbeiter in der Vergangenheit besser hätte auslasten müssen, oder dass er einige Vorgänge besser in einer anderen Reihenfolge hätte planen sollen, kann er diese Fehler korrigieren und die Simulation erneut von Beginn an ausführen. Da sich die Simulation bei mehrfachem Ausführen jeweils gleich verhält, ist das Springen zu einem bestimmten Punkt in der Vergangenheit als zusätzliche Option nicht notwendig. Statt dessen kann die Simulation auf den Anfang zurückgesetzt und dann erneut bis zu dem gewünschten Zeitpunkt ausgeführt werden.

Personalmanagement

Neben der Reihenfolge der Vorgänge und ihrer Dauer, hat auch die Auswahl der Mitarbeiter einen erheblichen Einfluss auf das Ergebnis. Jedem einzelnen

Mitarbeiter kann über eine Vielzahl verschiedener Parameter ein ganz individueller Charakter zugewiesen werden. Dieser Charakter bestimmt das Arbeits- und Lernverhalten, die soziale Interaktion mit der Umwelt und die Fähigkeiten in Bezug auf das Arbeitsfeld eines Mitarbeiters. Die Werte der Parameter können entweder im Szenario definiert, oder von der Simulation während des Spiels zufällig gewählt werden.

Im realen Leben sind diese Parameter nur schwer messbar und selten in irgendeiner Art direkt sichtbar. Ein Projektleiter muss deshalb seine Mitarbeiter kennen und ein Gefühl dafür entwickeln, wie er sie entsprechend ihres Charakters am besten einsetzt. Da in einem Spiel diese Art des Kennenlernens nur schwer umsetzbar ist, müssen wir dem Spieler entsprechende Hilfestellungen geben. Das können beispielsweise zusätzliche beschreibende Texte sein, die im Szenario enthalten sind. Eine weitere Möglichkeit ist das Anzeigen einiger ausgewählter Eigenschaften der Mitarbeiter. Für unser Spiel haben wir eine Kombination aus beiden Wegen gewählt. Da die Fähigkeiten der Mitarbeiter einen hohen Einfluss auf die Produktivität und die Qualität der Arbeit haben, kann zu jedem Mitarbeiter das Fähigkeitsprofil angeschaut werden.

Abbildung 5 zeigt die Mitarbeiterverwaltung. Bei Auswahl eines Mitarbeiters werden rechts dessen Fähigkeiten zu den einzelnen Aufgaben und den produktspezifisch relevanten Domänen als Balkendiagramme angezeigt. Zusätzlich wird rechts oben durch ein Icon ein Hinweis auf den gesundheitlichen Zustand des Mitarbeiters eingeblendet. So kann der Spieler erkennen, wie fit seine Mitarbeiter sind, und diese Information bei der Planung der Arbeitspakete berücksichtigen.

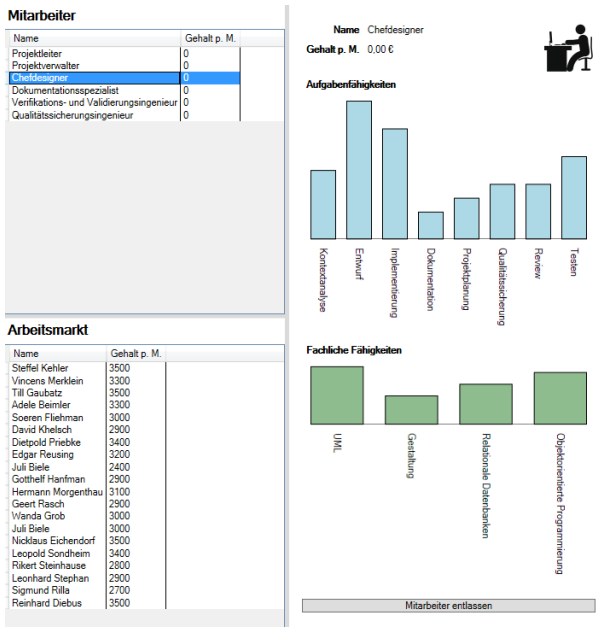


Abbildung 5: Mitarbeiterübersicht

Wird ein Bewerber auf dem Arbeitsmarkt ausgewählt, werden statt den realen Werten die Werte aus dessen Portfolio angezeigt. Das Portfolio eines Mitarbeiters entspricht einer Bewerbung, in der der Mitarbeiter sich selbst beschreibt. Diese Beschreibung kann je nach Charakter des Mitarbeiters entsprechend von den tatsächlichen Werten abweichen. So kann ein Mitarbeiter nicht in der Lage sein, seine Fähigkeiten entsprechend gut einzuschätzen, oder er stellt sich absichtlich besser oder schlechter dar, als er tatsächlich ist. Diese Diskrepanz zeigt sich erst nach der Einstellung des Mitarbeiters. Da eine Einstellung immer mit einem finanziellen Aufwand und einer zeitlichen Verzögerung verbunden ist, kann der Spieler nicht einfach alle Mitarbeiter ausprobieren. Es besteht deshalb – wie auch in der Realität – bei der Einstellung eines neuen Mitarbeiters immer ein gewisses Risiko, einen ungeeigneten Mitarbeiter einzustellen.

Budget und Finanzen

Ein relevanter Teilaspekt, der zunächst nichts mit der Softwareentwicklung an sich zu tun hat, aber in nahezu jedem Projekt berücksichtigt werden muss, sind die Projektfinanzen. Vor allem bei Szenarien, in denen der Spieler seine Mitarbeiter selbst über den Arbeitsmarkt bezieht, sind die dadurch entstehenden Kosten relevant und wirken sich teils erheblich auf den Projekterfolg aus.

Jeder Mitarbeiter kostet Geld. Wie viel wird entweder im Szenario festgelegt oder durch den Arbeitsmarkt definiert. Bessere Mitarbeiter kosten tendenziell mehr als Mitarbeiter mit geringeren Fähigkeiten. Es gibt dabei allerdings auch Ausnahmen. Der Spieler muss lernen, die Fähigkeiten der Kandidaten in Bezug auf sein Projekt zu beurteilen, um so die richtigen Mitarbeiter auswählen zu können. Dabei muss er berücksichtigen, welche Vorteile ein Mitarbeiter für das Projekt hat, aber auch welche Kosten dabei entstehen. In manchen Fällen kann es sinnvoll sein, zwei günstigere Mitarbeiter einzustellen als einen teureren, oder umgekehrt. Der Spieler muss dabei auch berücksichtigen, dass durch die Kommunikation zwischen den Mitarbeitern Arbeitsleistung verloren gehen kann.

Mitarbeiter, die nicht ausgelastet sind, verursachen trotzdem Kosten. Daher muss der Spieler gut überlegen, wann er zusätzliche Mitarbeiter einstellt und wann er Mitarbeiter entlässt. Sowohl Einstellung als auch Entlassungen verursachen zusätzliche Kosten.

Neben dem Gehalt der Mitarbeiter, das sich von Mitarbeiter zu Mitarbeiter unterscheiden kann und mitunter von deren Tagesarbeitszeit abhängig ist, haben Mitarbeiter auch davon unabhängige, laufende Fixkosten. Zusätzlich gibt es weitere, laufende Fixkosten für das Projekt an sich, z.B. für Büroräume, Softwarelizenzen und Büromaterial. Zusätzliche, einmalige Kosten, ebenso wie zusätzliche Einnahmen, können im Szenario festgelegt werden. Zusätzliche

Einnahmen können beispielsweise die Anschubfinanzierung des Projekts oder regelmäßige Zahlungen des Kunden sein. Alle Kosten und Einnahmen kann der Spieler über den Budgetdialog einsehen und mitverfolgen (siehe Abbildung 6).

Abbildung 6: Budgetdialog

Ein negatives Budget kostet zusätzliches Geld für die dadurch fälligen Zinsen. Der Spieler sollte deshalb möglichst gut mit den ihm zur Verfügung stehenden Mitteln haushalten. Finanzielle Gewinne und Verluste haben keine direkte Auswirkung auf das Projekt an sich. Sie gehen aber in die abschließende Bewertung des Projekts ein.

Wer gewinnt? – Score und Projektbericht

Der Spieler entscheidet selbst, wann das Projekt endet. Typischerweise wurde entweder das Produkt vollständig und in einer annehmbaren Qualität entwickelt, oder das im Szenario festgelegte Enddatum für das Projekt wurde überschritten. Das Spiel selbst kann trotz Projektende beliebig fortgesetzt werden. So kann die Projektlaufzeit bei Bedarf auch überschritten werden.

Wichtig für den Lernerfolg ist ein konstantes Feedback an den Spieler. Dafür verwenden wir die Projektbewertung. Sie besteht aus einem Score und einem zusätzlichen kurzen Projektbericht, der die einzelnen Teilaspekte zusammenfasst und bewertet. Dieser, in Abbildung 7 dargestellte Bericht, kann vom Spieler jederzeit eingesehen werden. So bekommt der Spieler auch schon während des Spiels eine Rückmeldung, wo sein Projekt steht.

Die Bewertung enthält die wesentlichen Projektbewertungskategorien: Vollständigkeit, Qualität, Kosten und Dauer. Vollständigkeit und Qualität können zu einer Produktbewertung zusammengefasst werden. Um einen, auch über mehrere Projekte hinweg vergleichbaren Wert für die einzelnen Aspekte zu erhalten,

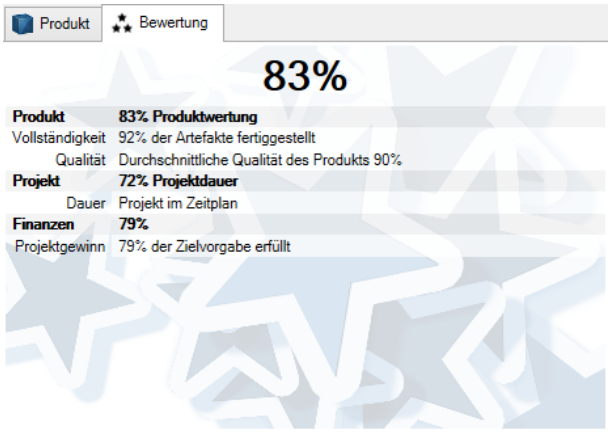


Abbildung 7: Score und Projektbericht

werden die Bewertungen in einem Prozentsatz ausgedrückt. Dabei steht 100% für ein optimales Ergebnis. In wenigen Fällen, wenn beispielsweise beim Projektgewinn ein Optimum aufgrund der nach oben offenen Skala nicht definiert werden kann, sind auch Werte größer 100% möglich.

Die Vollständigkeit c wird, wie in Formel (1) gezeigt, anhand des bearbeiteten Umfangs der einzelnen Artefakte und des Gesamtumfangs des Produkts berechnet.

Die Qualität des Produkts wird als die mit dem Umfang der Artefakte gewichtete Summe der einzelnen Artefakte berechnet (2). Jedes Artefakt besitzt dabei ein Qualitätslevel a_q zwischen 0 und 1. Dabei steht 1 für absolute Fehlerfreiheit bzw. ein optimales Ergebnis, 0 für ein vollständig falsches, bzw. aufgrund der hohen Fehlerzahl unbrauchbares Artefakt. Ein Qualitätswert von 1 bzw. 100% entspricht damit einem Produkt mit maximaler Qualität.

Die Produktwertung ist die Kombination aus Qualität und Vollständigkeit der Artefakte (3). Da die Vollständigkeit absolut in Arbeitsstunden angegeben ist, gehen die Artefaktattribute entsprechend ihres Umfangs gewichtet in die Produktwertung ein. Kleine Artefakte haben daher eine geringere Auswirkung auf die Produktwertung als Artefakte mit größerem Umfang.

Durch das Szenario wird für jedes Projekt eine Projektlaufzeit mittels Starttermin und Endtermin vorgegeben. Eine Überschreitung des Termins ist möglich, wirkt sich aber negativ auf die Wertung aus. Die Projektdauer berechnet sich wie in (4) gezeigt, aus der vorgegebenen und der tatsächlichen Projektlaufzeit.

Die Kostenwertung wird, wie in (5) gezeigt, anhand des aktuellen Kontostands berechnet. Dabei wird zusätzlich der bereits umgesetzte Produktumfang berücksichtigt. Das verhindert eine hohe Wertung, die zu Beginn des Projekts oder durch das Ansammeln von Finanzmitteln, ohne die Investition in das Produkt entstehen könnte. Entsprechend der Annahme, dass ein Produkt mit einem höheren Umfang auch einen

$$c = \frac{\sum_{a \in A} a_c}{\sum_{a \in A} a_s} \quad (1)$$

$$Qualität = \frac{\sum_{a \in A} a_q \cdot a_s}{\sum_{a \in A} a_s} \quad (2)$$

$$Produkt = \frac{\sum_{a \in A} a_c \cdot a_q}{\sum_{a \in A} a_s} \quad (3)$$

$$Dauer = \frac{t_{Heute} - t_{Start}}{t_{Ende} - t_{Start}} \quad (4)$$

$$Kosten = max\left(0, 0.5 + \frac{k \cdot b}{\sum_{a \in A} a_s} \cdot \left\{ \begin{array}{ll} c & b > 0 \\ 1 & \text{sonst} \end{array} \right)\right. \quad (5)$$

$$Score = \frac{Produkt \cdot 0.5}{max(1, Dauer)} + Kosten \cdot 0.5 \quad (6)$$

A : Menge aller Produktartefakte
a_c : Fortschritt von Artefakt a in Arbeitsstunden
a_q : Qualität des Artefakts $\in$ [0,1]
a_s : Umfang von Artefakt a in Arbeitsstunden
t_{Heute} : Aktuelles Datum der Simulation
t_{Start} : Datum des Projektbeginns
t_{Ende} : Datum des vorgegebenen Projektendes
b : aktueller Kontostand in €
k : Faktor entsprechend der Gewinnerwartung

entsprechend höheren Umsatz und Gewinn bedeutet, wird die Kostenwertung über den Produktumfang normiert.

Um einen relativen Wert für die Wertung zu bekommen, muss die Gewinnerwartung k berücksichtigt werden. Diese ist im Szenario definiert und hängt maßgeblich von den Entwicklungskosten und den Zahlungen des Kunden ab. Ein kostenneutrales Projekt wird mit 50% bewertet. Ein Projekt, das die Gewinnerwartung erfüllt mit 100%. Projekte, die Verluste verursachen, erhalten eine Wertung unter 50%.

Die Gesamtwertung (6) setzt sich aus den oben beschriebenen Teilwertungen zusammen. Dabei enthält die Produktwertung die Vollständigkeit und Qualität, die Kostenwertung die Vollständigkeit und den Projektgewinn bzw. -verlust. Die Kostenwertung enthält indirekt die Projektdauer, da eine längere Projektlaufzeit aufgrund der laufenden Kosten den Gewinn reduziert. Produkt- und Kostenwertung gehen zu gleichen Teilen in die Gesamtwertung ein. Da in der Produktwertung die Projektlaufzeit nicht enthalten ist, muss diese hier besonders berücksichtigt werden. Dazu wird die Produktwertung durch die zur veranschlagten Projektlaufzeit relativen tatsächlichen Laufzeit geteilt.

Um zu verhindern, dass Anomalien in der Bewertung entstehen, weil ein Projekt sehr früh beendet wird, wird eine Projektlaufzeit unter der im Szenario veranschlagten Zeit nicht zusätzlich belohnt. Da durch ein frühes Beenden des Projekts die Kostenwertung verbessert werden kann, geht dieser Aspekt trotzdem indirekt in die Gesamtwertung ein.

Es besteht die Möglichkeit, Szenarien ohne Budgetaspekte zu definieren, wenn diese für das Lernziel nicht relevant sind. In diesem Fall entfällt die Kostenwertung und die Gesamtwertung entspricht nur dem Quotienten aus Produktwertung und Projektdauer.

Beispielszenario: Softwaregrundprojekt

Ein wesentlicher Meilenstein der Informatikausbildung an der Universität Ulm ist das Softwaregrundprojekt. Hier kommen die Studierenden zum ersten Mal mit der Softwaretechnik in Berührung. Während sie zu Beginn des Studiums lediglich kleinere Übungsaufgaben implementiert haben, müssen sie im Softwaregrundprojekt zum ersten Mal ein vollständiges System nach ausgewählten Prinzipien der Softwaretechnik entwickeln. Dieses Praktikum geht über zwei Semester und wird von einer Vorlesung zur Softwaretechnik begleitet.

Wir verwenden im Softwaregrundprojekt eine angepasste Form der Fusion-Methode nach Coleman et al. (Coleman u. a., 1994) mit einer UML-Erweiterung angelehnt an die Ideen in (Bittner, 2006). Da die Studierenden noch keinerlei Erfahrung mit Fusion oder anderen Methoden der Softwaretechnik gemacht haben, fällt es ihnen anfänglich oft schwer, sich in der Vielzahl der unterschiedlichen Aktivitäten und zu erstellenden Artefakte zurechtzufinden.

Um den Studierenden möglichst früh einen Überblick über die wichtigen Elemente des Projekts und deren Zusammenhänge zu ermöglichen, haben wir das Softwaregrundprojekt als Szenario für unser Spiel modelliert. Während im Praktikum selbst ein konkretes System entwickelt wird, ist das Produkt im Spiel sehr abstrakt gehalten und von der konkreten Aufgabenstellung des Praktikums unabhängig.

Wenn im Folgenden vom *Softwaregrundprojekt* gesprochen wird, ist immer die tatsächliche Lehrveranstaltung gemeint. Dagegen beschreiben wir mit *Spiel* oder *Szenario* das simulierte Projekt in der Spielumgebung. Sowohl die Aktivitäten, als auch die Artefakte des Szenarios leiten sich direkt aus der Aufgabenstellung und den zu erstellenden Teilergebnissen im Softwaregrundprojekt ab.

Vorgehensmodell

Das Vorgehensmodell definiert die Rollen, Aktivitäten und Artefakttypen im Projekt. Da das Softwaregrundprojekt so ausgelegt ist, dass jeder der Studierenden jede der unterschiedlichen Aktivitäten selbst mindestens einmal durchgeführt hat, benötigen wir kein besonderes Rollenkonzept, welches die einzelnen Mitarbeiter auf ihre Aufgaben einschränkt.

Die Studierenden müssen im Laufe des Projekts sieben unterschiedliche Aktivitäten ausführen:

- **Kontextanalyse**: Zu Beginn des Projekts muss anhand von Produktskizzen und Kundengesprächen herausgefunden werden, was der Kunde für ein System wünscht. Diese Erkenntnisse müssen geeignet dokumentiert bzw. modelliert werden. Die Kontextanalyse bearbeitet Artefakte des Typs *Anforderungsdokument* und *UI Spezifikation*.
- **Entwurf**: Mittels der Entwurfsaktivitäten wird auf Basis der Dokumente, die während der Kontextanalyse erstellt wurden, das Architekturdesign des Systems entwickelt. Beim Entwurf werden Artefakte des Typs *Entwurfsdokument* bearbeitet.
- **Implementierung**: Die Implementierungsaktivität beschreibt sowohl das Erstellen des Quellcodes, als auch dessen Dokumentation. Dabei werden Artefakte des Typs *Implementierung* bearbeitet.
- **Dokumentation**: Neben der inhärenten Dokumentation der einzelnen Artefakte, ist für viele Bereiche eine zusätzliche Dokumentation, wie beispielsweise ein Benutzerhandbuch, notwendig. Dabei werden Artefakte des Typs *Dokumentation* erstellt.
- **Projektplanung**: Diese Aktivität befasst sich mit allem, was zur Planung und Steuerung des Projekts gehört. Der dazugehörige Artefakttyp ist *Controlling*.
- **Qualitätssicherung**: Vorbereitende und begleitende Maßnahmen zur Qualitätssicherung. Erstellt werden Artefakte des Typs *Qualitätssicherung*.
- **Testen**: Dient zur Verbesserung der Qualität von Artefakten des Typs *Implementierung*.

Alle genannten Aktivitäten, mit Ausnahme von *Testen*, sind Entwicklungsaktivitäten, die dazu dienen, verschiedene Artefakte zu erstellen. Testen ist eine Qualitätssicherungsaktivität, die dazu dient, die Qualität von bestehenden Artefakten zu verbessern, indem die darin enthaltenen Fehler aufgedeckt und behoben werden.

Produkt

Das zu entwickelnde Produkt besteht aus ca. 60 Artefakten. Jedes dieser Artefakte ist einem der Artefakttypen zugeordnet. Die einzelnen Artefakte haben einen individuellen Schwierigkeitsgrad und stellen unterschiedliche Anforderungen an die fachlichen Fähigkeiten der Mitarbeiter. Der Umfang der einzelnen Artefakte ist ebenfalls unterschiedlich.

Den Artefakten sind teilweise ein oder mehrere der vier Fachbereiche *UML*, *Gestaltung*, *Relationale Datenbanken* und *Objektorientierte Programmierung* als Schwerpunkt zugeordnet. Je besser sich diese Fachbereiche mit den entsprechenden Fähigkeiten der zuständigen Mitarbeiter decken, umso besser sind die Ergebnisse der Aktivitäten.

Alle im Szenario definierten Artefakte müssen von den Studierenden später auch im Softwaregrundprojekt erstellt werden. Sie bauen zum Teil aufeinander auf bzw. beeinflussen sich gegenseitig. Diese Abhängigkeiten sind auch im Szenario als Voraussetzungen und Einflüsse modelliert worden. So beeinflussen beispielsweise die Programmierkonventionen die Qualität der Implementierung und Tests können nicht ohne das vorherige Erstellen der entsprechenden Testpläne durchgeführt werden.

Das Produkt besteht aus folgenden Artefakten:

- **Anforderungsdokumente**: Überblick, Fachwissen, Anwendungsfälle, Szenarien, Systemaufgaben, Nichtfunktionale Anforderungen, Domänenmodell, Schnittstellenmodell und Pflichtenheft
- **UI Spezifikationen**: Dialogstruktur, Dialoggestaltung und Nutzungskonzept
- **Entwurfsdokumente**: Datenbanklayout, Kommunikationsmodell, Klassenmodell und Methodenbeschreibung
- **Implementierung**: Systemkern, Tools, Datenbankanbindung, Übergreifende UI Konzepte und fünf Komponenten mit jeweils einem Artefakt für Benutzeroberfläche und Logik.
- **Controlling**: Projektplan und Rollenverteilung
- **Qualitätssicherung**: Allgemeine Testrichtlinien und jeweils ein Testplan zu jedem Implementierungsartefakt.
- **Dokumentation**: Programmierkonventionen, Architekturbeschreibung, Komponentendiagramm, Beschreibung des Gesamtkonzepts, Produkt- und Lizenzlisten, Datenbankbeschreibung, Installationsanleitung, Benutzerdokumentation und Projekttagebuch.

Mitarbeiter

Das Softwaregrundprojekt wird in Teams zu jeweils sechs Studenten durchgeführt. Zu Beginn arbeiten dabei alle gemeinsam an der Kontextanalyse. Nachdem diese Phase mit der Erstellung des Pflichtenhefts abgeschlossen wurde, werden die Verantwortlichkeiten für die folgenden Aufgaben unter den Teammitgliedern aufgeteilt. Es gilt aber weiterhin, dass sich jedes Teammitglied an allen Aufgaben beteiligen soll, um in jeden Bereich Einblick zu erhalten. Lediglich die Schwerpunkte der einzelnen Teammitglieder können sich unterscheiden.

Die Verantwortlichkeitsbereiche sind *Systemarchitektur*, *Qualitätssicherung*, *Verifikation und Validierung*, *Dokumentation*, *Infrastruktur und Projektverwaltung* und *Projektmanagement*. Die Implementierung ist kein eigener Bereich, da sich hier alle Studenten in gleichem Umfang beteiligen sollen.

Im Szenario ist jeweils ein Mitarbeiter definiert, der von seinem Fähigkeitsprofil gut zu einem der Bereiche passt. Der Bereich Projektmanagement wird vom Spieler selbst übernommen und bedarf deshalb keinem simulierten Mitarbeiter. Die Projektverwaltung lässt sich im Szenario nicht sinnvoll umsetzen, da es sich hierbei um eine übergeordnete Aufgabe handelt. Daher wurde Projektverwalter und Projektleiter jeweils

ein Fähigkeitsprofil zugewiesen, das die der anderen Mitarbeiter möglichst gut ergänzt. Dadurch steht dem Spieler ein ausgewogenes Team zur Verfügung.

Da das Softwaregrundprojekt mit einem festen Projektteam und ohne Finanzplanung durchgeführt wird, wurden die Budgetplanung sowie der Arbeitsmarkt deaktiviert.

Dieses Szenario stellt die Grundlage für unsere Evaluation dar.

Evaluation

Um den Lerneffekt, die Akzeptanz und die Bedienbarkeit des Spiels zu testen, haben wir eine Evaluation in Form einer Onlinebefragung unter den Spielteilnehmern durchgeführt.

Da wir das Spiel bisher nur einer kleinen Gruppe von 14 Studierenden zugänglich gemacht haben, sind die quantitativen Ergebnisse dieser Befragung nicht ausreichend belastbar und wurden deshalb durch eine qualitative Evaluation in Form von detailliertem persönlichem Feedback ergänzt.

Da die quantitative Auswertung der Ergebnisse der Onlinebefragung sowohl mit unseren Erwartungen, als auch mit den Ergebnissen des persönlichen Feedbacks nahezu übereinstimmt, sind wir momentan dabei, die Evaluation auf eine weitere Gruppe mit ca. 150 Studierenden auszuweiten. Ziel dabei ist es, die vorliegenden Ergebnisse auch quantitativ bestätigen zu können.

Der verwendete Fragebogen enthält 30 Fragen, die in sieben Blöcke aufgeteilt sind:

1. Fragen zur Erfassung der Rahmensituation. Dazu wurden Fragen zum allgemeinen Spielverhalten und zum Umgang mit komplexen Problemen gestellt.
2. Fragen zum Spielerlebnis. Dazu gehören beispielsweise Spielzeit, Spielspaß und Wiederspielwert des Spiels.
3. Fragen zur Erfassung des Lernfortschritts in Bezug auf das Werkzeug MS Project. Dazu wurde unter anderem der diesbezügliche Kenntnisstand vor und nach dem Spiel erfasst.
4. Fragen zur Erfassung des Lernfortschritts in Bezug auf das Vorgehensmodell. Dazu wurde wie bei 3. der entsprechende Kenntnisstand vor und nach dem Spiel erfasst. Anschließend wurden Fragen gestellt, bei denen die Spieler ihren Kenntnisstand in Bezug auf das im Szenario verwendete Vorgehensmodell einschätzen sollten.
5. Analog zu 4. wurde der Lernfortschritt in Bezug auf das Produkt, d.h. die Dokumente und Modelle, die während des Softwaregrundprojekts erstellt werden müssen, evaluiert.
6. Fragen zur Erfassung allgemeiner Lernerfolge im Projektmanagementbereich.
7. Abschließende Fragen zur Darstellung der Informationen im Spiel und der Bedienbarkeit von Spiel und MS Project.

Im Folgenden liefern wir eine kurze Zusammenfassung der Ergebnisse der Befragung.

Der allgemeine Spielspaß wurde von den Teilnehmern als eher gering eingestuft. Dieses Ergebnis ist nicht weiter überraschend, da sich das Spiel auf die reinen Lernaspekte beschränkt und keine zusätzlichen Elemente eingebaut wurden, die dem Spielspaß dienen. Der Aspekt des Lernens steht bei diesem Spiel eindeutig im Vordergrund und das wird von den Studierenden auch so erkannt. Trotzdem würden die meisten Teilnehmer das Spiel mit einem anderen Szenario erneut spielen und es auch ihren Kommilitonen weiterempfehlen. Das spricht in unseren Augen für die Akzeptanz des Spiels als Lernmittel und lässt einen gewissen Wiederspielwert erkennen.

Anhand der Fragen zu MS Project konnten wir feststellen, dass alle Spieler ihre Fähigkeiten in Bezug auf das Werkzeug nach dem Spiel höher einschätzen als vor dem Spiel. Daraus schließen wir, dass die Benutzung von MS Project im Rahmen des Spiels auch einen Trainingseffekt auf den Umgang mit diesem und ähnlichen Werkzeugen hat. Das wird auch von den Spielern selbst so gesehen. Alle haben angegeben, durch das Spiel im Umgang mit MS Project etwas gelernt zu haben, wenn auch unterschiedlich viel. Alle Teilnehmer trauen sich jetzt zu, ein einfaches Projekt mit MS Project zu managen. Auch diejenigen, die angegeben haben, vor dem Spiel mit MS Project nicht zurechtgekommen zu sein.

Ein ähnliches Ergebnis konnten wir bei den allgemeinen Fähigkeiten ein Projekt zu planen beobachten. Auch hier trauen sich jetzt alle Spieler zu, eigenständig ein Projekt zu planen. Allerdings haben die meisten angegeben, diese Fähigkeit auch schon vor dem Spiel besessen zu haben. Trotzdem haben alle Spieler angegeben, durch das Spiel ihre Projektmanagementfähigkeiten verbessert zu haben.

Im Bezug auf die Aktivitäten und Dokumente, die im Softwaregrundprojekt benötigt werden, haben beinahe alle Spieler nach Abschluss des Spiels angegeben, die Aktivitäten und Artefakte benennen, erklären und in ihren Kontext einordnen zu können.

Mit der Bedienung des Spiels sind die Teilnehmer gut zurechtgekommen. Für die Darstellung des Szenarios gab es einige Anregungen zur Verbesserung der Übersichtlichkeit der Zusammenhänge von Aktivitäten und Artefakten. Die Meinungen zur Bedienbarkeit von MS Project, unabhängig des Spiels, waren sehr unterschiedlich und gingen von *problemlos* bis hin zu *nur schlecht bedienbar*.

Die Ergebnisse zeigen uns, dass die grundsätzliche Konzeption des Spiels erfolgversprechend ist. Es ist notwendig, einige kleinere Anpassungen vorzunehmen, um das Spiel und dessen Bedienbarkeit weiter

zu optimieren. Sollte eine umfangreichere Evaluation diese Ergebnisse bestätigen, ist geplant, das Spiel zusätzlich um neue Konzepte zu erweitern. Eines der Hauptziele wird es dann sein, den Spielspaß zu erhöhen.

Fazit und Ausblick

Die Ausbildung von Studierenden im Bereich des Projektmanagements stellt für die Lehrenden eine große Herausforderung dar, da die notwendige praktische Erfahrung nur durch geeignete eigenständige Projektarbeit gewonnen werden kann.

Mit unserem Planspiel haben wir eine Möglichkeit gefunden, praktische Erfahrung spielerisch zu gewinnen. Ziel dabei war es, eine möglichst natürliche Spielumgebung zu schaffen, die es dem Spieler ermöglicht, neben den Erfahrungen im Bereich des Projektmanagements gleichzeitig den Umgang mit den dazu benötigten Werkzeugen zu üben. Dazu haben wir ein Simulationsframework als AddIn in MS Project integriert, so dass der Spieler die mit MS Project geplanten Projektvorgänge direkt simulieren und dadurch die Ergebnisse seiner Planung sofort sehen kann.

Die von uns durchgeführte Evaluation hat gezeigt, dass wir mit dieser Art von Lernspiel einen erfolgversprechenden Weg gewählt haben. Um das volle Potential des Ansatzes nutzen zu können, muss diese Idee noch weiter verfeinert und ausgebaut werden.

Als erster Schritt soll das Feedback, das der Spieler durch das Spiel erhält, erweitert werden. Um seine Mitarbeiter besser beobachten, einschätzen und steuern zu können, ist es sinnvoll, dem Spieler weitere Informationen über den Zustand der Mitarbeiter und dem Team im Gesamten zu geben. Dazu gehören zum Beispiel die Motivation der Mitarbeiter, das soziale Gefüge im Team, Vorlieben und Abneigungen der einzelnen Mitarbeiter für bestimmte Aufgaben, sowie diverse weiche Eigenschaften wie Lerngeschwindigkeit und Kommunikationsfähigkeit. All diese Parameter werden momentan zwar von der Simulation berücksichtigt, dem Spieler aber nicht mitgeteilt. Durch eine geeignete Aufbereitung dieser Informationen und einer Integration in die Spielumgebung wird das Spiel für den Spieler realistischer und besser beherrschbar.

Ein ähnlicher Ansatz gilt für die Darstellung des Produkts, den Anforderungen für die Erstellung der einzelnen Artefakte, sowie deren Abhängigkeiten und Einflüsse untereinander. Hier kann vor allem eine geeignete Visualisierung helfen, das zu entwickelnde Produkt besser zu verstehen und damit auch das Projekt besser steuern zu können.

Die Berechnung der Gesamtwertung des Projekts berücksichtigt momentan lediglich den Umfang der einzelnen Produktartefakte. In der Realität sind nicht alle Artefakte gleich wichtig für den Erfolg eines Projekts. Die Wichtigkeit eines Artefakts ist nicht ausschließlich an dessen Umfang zu erkennen, sondern setzt zusätzliche Informationen über die Bedeutung bzw. Aufgabe des Artefakts im Produkt voraus. Ein möglicher Ansatz für eine Verbesserung der Berechnung der Score ist es, einen entsprechenden zusätzlichen Gewichtungsfaktor für die einzelnen Artefakte einzuführen. Eine mögliche weitere Verfeinerung dieses Ansatzes ist es, die Qualität und den Umfang getrennt zu gewichten. So kann bei für das Produkt kritischen Artefakten ein höheres Augenmerk auf die Qualität gefordert werden.

Am Ende des Spiels soll für den Spieler, neben dem kurzen Übersichtsbericht, ein zusätzlicher Projektbericht über den Verlauf seines Projekts erstellt werden. Je nach Szenario kann dieser dem Spieler auch schon während des Spiels zugänglich gemacht werden. Ein solcher Bericht kann den zeitlichen Fortschritts- und Qualitätsverlauf der einzelnen Artefakte enthalten. Ebenso hilfreich ist eine Verfolgung der Eigenschaften der einzelnen Mitarbeiter. So kann schnell erkannt werden, wie sich Projektentscheidungen auf beispielsweise die Motivation, die Fitness oder die Fähigkeiten der Mitarbeiter ausgewirkt haben. All diese Informationen müssen geeignet aufbereitet und dargestellt werden.

Bislang unterstützt das Spiel ausschließlich Szenarien, die auf linearen Prozessmodellen basieren. Grund dafür ist die, über die Szenarien definierte, statische Konfiguration des Produkts, bestehend aus seinen einzelnen Artefakten. Bei iterativen und agilen Prozessmodellen ist diese Aufteilung zu Beginn des Projekts noch nicht vollständig bekannt. Um nichtlineare Prozessmodelle zu unterstützen, muss der Spieler während des Projekts entscheidet können, welche neuen Artefakte zu erstellen sind oder welche bestehenden Artefakte verworfen werden sollen. Damit gibt er vor, wie er die Entwicklung des Produkts strukturieren möchte. Das kann beispielsweise durch das Erstellen einer neuen Iteration bei iterativen Prozessmodellen, oder innerhalb der Planung eines neuen Sprints bei agilen Prozessmodellen wie SCRUM geschehen.

In wie weit sich eine solche zusätzliche Funktionalität in das Spiel integrieren lässt, haben wir bislang noch nicht getestet. Da das verwendete Simulationsframework die Möglichkeit bietet, die Struktur des Produkts während der Simulation zu verändern, sind zumindest die technischen Grundlagen dafür vorhanden.

Die Evaluation hat uns gezeigt, dass noch Potential zur Steigerung des Spielspaßes vorhanden ist. Beim Design unseres Testszenarios haben wir darauf bewusst keinen Wert gelegt. Langfristig macht es aber Sinn ein Spiel zu entwickeln, das dem Spieler auch

Spaß macht. Ein möglicher Ansatzpunkt dafür ist eine detailliertere Gestaltung der Szenarien. Diese können mit zusätzlichen Spielelementen, wie beispielsweise zufälligen Spielereignissen erweitert werden. Eine weitere Möglichkeit das Spiel abwechslungsreicher zu gestalten ist es, das Szenario um eine Storyline zu ergänzen und diese medial mit Texten, Grafiken und Videos ansprechend zu gestalten. Diese Spielelemente zielen primär darauf ab, das Spiel abwechslungsreicher und spannender zu machen.

Die Erfahrungen, die wir mit diesem Projekt gemacht haben, zeigen uns, dass es sinnvoll ist, bei der Entwicklung von Lernspielen eine möglichst realitätsnahe Spielumgebung zu schaffen. Es hat sich herausgestellt, dass sich dies durch das Einbeziehen der Werkzeuge, die in der realen Entwicklung verwendet werden, umsetzen lässt. Da diese Werkzeuge häufig eine gute Erweiterbarkeit mit entsprechenden Schnittstellen aufweisen, lässt sich eine Integration häufig ohne großen Aufwand realisieren.

In unserem Spiel haben wir uns auf den Aspekt des Projektmanagements beschränkt. Ein interessanter Ansatz für weitere Arbeiten ist es, andere Aspekte des Software Engineering in ähnlicher Weise in die passenden Werkzeuge zu integrieren, und diese auf einer höheren Ebene zu einem gemeinsamen Softwareprojekt zusammenzufügen. So können die Studierenden alle relevanten Aspekte im Bereich des Software Engineering an einem einzigen durchgehenden Projekt praktisch üben.

Literatur

[Bittner 2006] BITTNER, M.: *Enhancing the Fusion Method to Fusion B: Requirements Engineering and Formal Specification*, Technischen Universität Berlin, Diss., 2006

[Coleman u. a. 1994] COLEMAN, Derek ; ARNOLD, Patrick ; BODOFF, Stephanie ; DOLLIN, Chris ; GILCHRIST, Helena ; HAYES, Fiona ; JEREMAES, Paul: *Object-oriented Development: The Fusion Method*. Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1994

[Jain u. Boehm 2006] JAIN, Apurva ; BOEHM, Barry W.: SimVBSE: Developing a Game for Value-Based Software Engineering. In: *CSEE&T*, IEEE Computer Society, 2006, S. 103–114

[Ludewig u. a. 1992] LUDEWIG, J. ; BASSLER, T. ; DEININGER, M. ; SCHNEIDER, K. ; SCHWILLE, J.: SESAM-simulating software projects. In: *Software Engineering and Knowledge Engineering, 1992. Proceedings., Fourth International Conference on*, 1992, S. 608–615

[Microsoft Corporation 2013] MICROSOFT CORPORATION: *Microsoft Project*. Version 2013. 2013

[Nassal 2014] NASSAL, Alexander: A General Framework for Software Project Management Simulation Games. In: *Information Systems and Technologies (CISTI) 2014, 9th Iberian Conference on Information Systems and Technologies*, 2014, S. 321–325

[Navarro 2006] NAVARRO, Emily: *SimSE: A Software Engineering Simulation Environment for Software Process Education*, University of California, Irvine, Diss., 2006

[de Oliveira Barros u. a. 2006] OLIVEIRA BARROS, Márcio de ; DANTAS, Alexandre R. ; VERONESE, Gustavo O. ; WERNER, Cláudia Maria L.: Model-driven Game Development: Experience and Model Enhancements in Software Project Management Education. In: *Software Process: Improvement and Practice* 11 (2006), Nr. 4, S. 411–421

[Pieper 2013] PIEPER, Jöran: Alles nur Spielerei? Neue Ansätze für digitales spielbasiertes Lernen von Softwareprozessen. In: SPILLNER, Andreas (Hrsg.) ; LICHTER, Horst (Hrsg.): *Tagungsband des 13. Workshops "Software Engineering im Unterricht der Hochschulen" (SEUH) 2013, Aachen*, CEUR Workshop Proceedings Vol. 956, 2013, 131–139

[Shaw u. Dermoudy 2005] SHAW, Katherine ; DERMOUDY, Julian R.: Engendering an Empathy for Software Engineering. In: YOUNG, Alison (Hrsg.) ; TOLHURST, Denise (Hrsg.): *ACE* Bd. 42, Australian Computer Society, 2005 (CRPIT), S. 135–144

[Ye u. a. 2007] YE, En ; LIU, Chang ; POLACK-WAHL, J.A.: Enhancing Software Engineering Education Using Teaching Aids in 3-D Online Virtual Worlds. In: *Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, 2007

Praktikum Automotive Software Engineering: Best Practices

Michael Uelschen, Hochschule Osnabrück

m.uelschen@hs-osnabrueck.de

Zusammenfassung

Die Entwicklung von Software für Steuergeräte (ECU, electronic control unit) im Fahrzeug unterliegt speziellen Randbedingungen und Vorgehensweisen, die üblicherweise nicht Bestandteil in der einführenden Lehre des Software-Engineering sind. Dieser Beitrag untersucht die Frage, inwieweit die spezifischen Ausprägungen des Automotive Software Engineering (ASE) in der Ausbildung Studierender innerhalb eines Moduls vermittelt werden können.

Der vorgestellte Ansatz verzichtet bewusst auf den vollständigen Durchlauf aller Phasen im Entwicklungszyklus und setzt stattdessen einen Schwerpunkt auf spezifische Vorgehensweisen und Methoden bei der Software-Entwicklung im automobilen Bereich.

Einleitung

In den folgenden Abschnitten wird zuerst in die Thematik der Steuergeräte-Entwicklung eingeführt und die Bedeutung der Ausbildung im Bereich ASE hervorgehoben. Es werden die Unterschiede zum generellen Software Engineering herausgestellt.

Aus der arbeitsteiligen Vorgehensweise in der Automobilindustrie leiten sich Lernziele in Form von Kompetenzen für ein, im weiteren Verlauf des Beitrags vorgestelltes, Praktikum an der Hochschule Osnabrück ab. Die Erfahrungen und Beobachtungen während des Praktikums für Mechatronik-Studierende fließen in Empfehlungen zur Gestaltung einer entsprechenden Lehreinheit ein.

Automotive Software Engineering

Der Begriff Automotive Software Engineering (Schäuffele und Zurawka, 2012) umfasst Methoden, Werkzeuge und Vorgehensprozesse bei der Entwicklung von Software für Steuergeräte im Fahrzeug.

Hierbei handelt es sich im Allgemeinen um eingebettete Systeme, die Echtzeitanforderungen unterliegen. Durch die software-gesteuerte Verknüpfung von Sensorik und Aktuatorik lassen sich vielfältige Steuerungs- und Regelungsaufgaben realisieren. Hierbei gewinnen insbesondere Fahrerassistenzsysteme wie Parkassistent, Spurwarner oder adaptive Lichtregelungen zunehmend an Bedeutung (Winner et al., 2012).

Neben dem Entwurf neuer und der Verbesserung vorhandener Fahrfunktionen ist die Integration von Infotainment und Multimedia-Anwendungen eine wachsende Aufgabe des ASE. An der Schnittstelle zwischen der klassischen Automobilelektronik auf der einen und der Konsumgüterindustrie auf der anderen Seite entstehen zahlreiche neue Herausforderungen, die als *Innovation Cycle Dilemma* bezeichnet werden (Lucke et al., 2007).

Aufgrund der vielfältigen Fahrerinformations- und –assistenzsysteme gewinnt die Entwicklung einer geeigneten Fahrer-Fahrzeug-Schnittstelle an Bedeutung, um Informationen situationsabhängig unter Minimierung möglicher Ablenkungen bereitzustellen. Die Entwicklung multimodaler Mensch-Maschine-Schnittstellen für den Einsatz im Fahrzeug ist Gegenstand aktueller Forschung und Entwicklung (Pfleging et al., 2014).

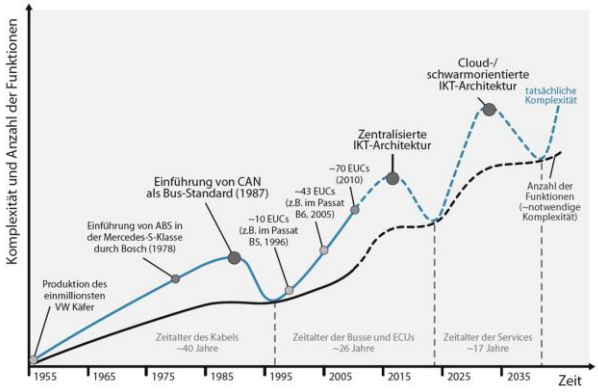


Abb. 1: Komplexität eines Fahrzeugs

Treiber und Herausforderungen

Broy zeichnet in (Broy, 2006) die evolutionären Schritte bei der Software-Entwicklung im Fahrzeug in den letzten 30 Jahren nach. Insbesondere durch die Einführung von fahrzeugspezifischen Netzwerken führen die bis zu 100 verteilten Steuergeräte innerhalb eines Automobils mehrere tausend

Funktionen aus. Moderne Fahrzeuge enthalten Software bis zu 100 Millionen Codezeilen (Mössinger, 2010).

Abbildung 1 zeigt die zunehmende Komplexität im Fahrzeug (Fortiss, 2011). Zusätzlich ist die tatsächliche Komplexität skizziert, die nicht aufgrund von Funktionalitäten, sondern durch Architektur- und Entwurfsentscheidungen entsteht. Ein Ende dieser Entwicklung ist zurzeit nicht absehbar. Hierbei sind die Herausforderungen im Bereich der Elektromobilität und der unterschiedlichen Stufen des autonomen Fahrens wesentliche technologische Treiber. Nach Schäuffele und Zurawka ist das „Automobil zum technisch komplexesten Konsumgut geworden".

In (Pretschner et al., 2007) sind die vielfältigen Herausforderungen des ASE zusammengefasst, die teilweise auch in anderen Domänen außerhalb der Automobilindustrie existieren.

Bedeutung

Aufgrund der zunehmenden Wertschöpfung im Fahrzeug durch Software (Weinmann, 2003), nimmt eine systematische Vorgehensweise bei der Entwicklung von Steuergeräten einen zunehmenden Stellenwert ein.

Die Automobilindustrie ist einer der wichtigsten Industriezweige in Deutschland. Die größten Automobilhersteller und –zulieferer haben hier ihren Standort (insbesondere aber nicht nur in Baden-Württemberg und in Bayern) und sind interessante Arbeitgeber für Ingenieure und Informatiker. Zusätzlich gibt es eine sehr große Anzahl von, i.allg. unbekannten, klein- und mittelständischen Unternehmen (*hidden champions*), die wichtige Komponenten entwickeln oder fertigen.

Eine Berücksichtigung in der Ausbildung Studierender reflektiert somit die industrielle Relevanz und entspricht somit dem Anspruch einer praxisnahen Berufsvorbereitung.

Neben der Entwicklung von Personen- und Lastkraftfahrzeugen entsteht inzwischen ein weiterer, eigenständiger Bereich innerhalb der ASE. Durch die fortschreitende Industrialisierung in der Landwirtschaft findet zunehmend die Entwicklung von Steuergeräten in der Landmaschinentechnik statt. Diese übernehmen nicht nur reine Fahrfunktionen, sondern automatisieren agrar-spezifische Prozesse.

Einige der weltgrößten Unternehmen der Landtechnik sind in der Region Osnabrück und Nordwestdeutschland ansässig, u.a. Krone, Amazone, Grimme und Claas.

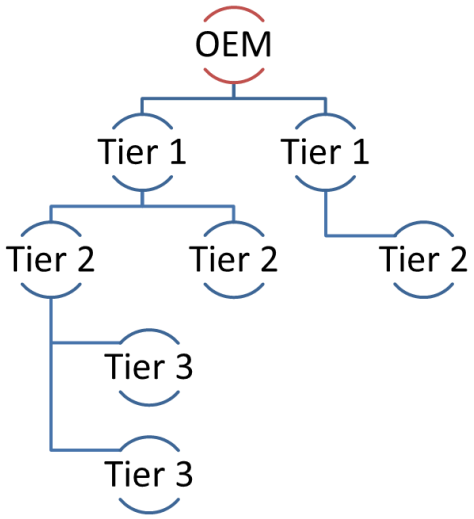


Abb. 2: Hierarchie der Automobilzulieferer

Arbeitsteilige Entwicklung

Die Entwicklung von Fahrzeugen, nicht nur der Steuergeräte, wird sehr stark durch Automobilzulieferer getrieben. Abbildung 2 zeigt hierbei eine dreistufige Hierarchie, bei der an der Spitze der Automobilhersteller (OEM, Original Equipment Manufacturer) als Auftraggeber steht. Die einzelnen Zulieferer ordnen sich in die Schichten Tier 1 bis Tier 3 ein, wobei Unternehmen, die der oberen beiden Schichten angehören, üblicherweise eine eigene Fertigung unterhalten. Entwicklungsdienstleister finden sich in der dritten Schicht wieder. Der OEM beauftragt einen Zulieferer, für eine bestimmte Funktionalität ein Steuergerät entsprechend den Vorgaben eines Lastenheftes zu entwickeln. Der zeitliche Entwicklungsrahmen wird durch den Termin des Neuanlaufs eines Fahrzeugs bestimmt. Aufgrund hoher Umsatzausfälle, die schnell die Millioneneurogrenze erreichen, ist eine Verschiebung von Neuanläufen ausgeschlossen, beispielsweise aufgrund fehlerhafter Software. Zur Absicherung der Terminplanung und der Risikominimierung setzt der OEM häufig auf einen zweiten Anbieter (*second source*).

Aufgrund dieses Vorgehens erfolgt die Entwicklung von Steuergeräten hochgradig parallel sowie räumlich getrennt bei unterschiedlichen Zulieferern. Da das neue Fahrzeug erst zu einem sehr späten Zeitpunkt im Entwicklungsprozess zur Verfügung steht, ist ein nicht unerheblicher Aufwand notwendig, um das umgebene System in frühen Phasen zu simulieren.

Vernetzung

Bei dieser sehr arbeitsteiligen Vorgehensweise in der Automobilindustrie ist es wichtig, dass die Schnittstellen der kommunizierenden Steuergeräte

zu einem sehr frühen Zeitpunkt im Entwicklungsprozess definiert sind. Die Kommunikation, der an die verschiedenen Netzwerke (LIN, CAN, FlexRay, Ethernet) angeschlossenen Komponenten, ist nachrichtenbasiert. Ein Nachrichtenkatalog beschreibt die nach außen sichtbare Funktionalität eines Netzknotens (Zimmermann und Schmidgall, 2014).

Die Nachrichten verändern den inneren Zustand eines Steuergeräts, so dass die Modellierung des Verhaltens neben der reinen Funktionsentwicklung einen wesentlichen Schwerpunkt bildet. Die Entwicklung gliedert sich hierbei in mehreren Stufen: zuerst wird das zu entwickelnde Steuergerät sowie die beteiligten Netzknoten simuliert. Danach wird das betreffende Steuergerät durch eine reale Komponente ersetzt (Restbussimulation). Im letzten Schritt wird das Steuergerät mit anderen realen Komponenten im Fahrzeug verbaut und anschließend erprobt.

Generelles Software Engineering

Im folgenden Abschnitt wird auf weitere Unterschiede eingegangen, die einen direkten Einfluss auf die Software-Entwicklung haben.

Grundsätzlich werden im ASE dieselben Aspekte und Phasen wie im generellen Software Engineering betrachtet (siehe beispielsweise in (Sommerville, 2012)). Viele Methoden lassen sich auf die ECU-Entwicklung übertragen.

Neben der Anwendungsdomäne, also den vielfältigen Fahrfunktionen, ergeben sich Unterschiede u.a. aufgrund weiterer Randbedingungen:

- Hoher Kosten- und Wettbewerbsdruck beeinflusst Entwurfsentscheidungen (beispielsweise geringe Prozessorleistung und Speicherkapazität).
- Unterschiedliche Klimazonen führen zu rauen Umgebungssituationen, auf die Hardware und Software reagieren müssen.
- Lange Produktzyklenzeiten stellen hohe Anforderungen an das Konfigurationsmanagement sowie an die Kompatibilität (siehe hierzu (Ritter, 2004)).

Es existiert eine Reihe von Branchenstandards, u.a. im Bereich der Entwicklungs- sowie Qualitätsverbesserungsprozesse, der Werkzeuge sowie der Architekturentwürfe:

- Die Software im Fahrzeug unterliegt zunehmend strengen Sicherheitsanforderungen, die konform zu ISO 26262 entwickelt werden muss (Gebhardt et al., 2013).
- Die Anstrengungen zur Qualitätsverbesserung orientieren sich an CMMI (Chrissis et al., 2011) oder Automotive SPICE (Höhn et al., 2009).
- Das AUTOSAR-Rahmenwerk definiert den grundsätzlichen Aufbau von automobilen Software-Anwendungen im Fahrzeug (Schäuffele und Zurawka, 2012).

Das Vorgehensmodell folgt in der Automobilindustrie dem V-Modell. Dieses wird um mehrere Iterationen oder Musterstände A bis D ergänzt, wobei das D-Muster den Serienstand und somit den Abschluss im Entwicklungsprozess darstellt. Agile Methoden werden bisher erst sehr wenig eingesetzt.

Die Software-Entwicklung im automobilen Bereich ist zunehmend modellbasiert und wird durch eine Reihe von proprietären Werkzeugen unterstützt.

Grundlagen

Für die ASE-Lehre bilden neben Programmierkenntnissen in der Sprache C/C++ die folgenden Themenbereiche die Grundlage:

- Regelungs- und Steuerungstechnik: Beschreibung kontinuierlicher Systeme
- Eingebettete Echtzeitsysteme
    - Ereignisbasierte und reaktive Systeme
    - Zustandsmaschinen beschreiben das dynamische Verhalten
    - Verwendung von Mikrocontrollern mit Peripherie
    - Weiche und harte Echtzeitanforderungen
    - Betriebssysteme: Berücksichtigung zeitlicher Anforderungen durch prioritätenbasiertes Scheduling
- Verteilte Systeme
    - Topologie/Gateway-Funktionalität
    - Echtzeitfähigkeit
    - Nachrichtenbasierte Kommunikation

Möglichkeiten in der Lehre

Einige Hochschulen in Deutschland bieten einen spezialisierten Master-Studiengang im Bereich Automotive Software Engineering an, u.a. die Universität Chemnitz und die Technische Universität München. Die Hochschule Landshut bietet einen Bachelor-Studiengang Automobilinformatik an. In vielen anderen fahrzeugspezifischen Studiengängen an Hochschulen in Deutschland ist die Belegung eines Moduls ASE verpflichtend oder kann aus dem Wahlpflichtkatalog ausgewählt werden. Auch fächerübergreifende Studiengänge bieten eine verstärkte Kompetenzbildung im Bereich Automotive Software an (beispielsweise Wirtschaft-

singenieurwesen Automotive an der Bergischen Universität Wuppertal).

Dieser Beitrag skizziert ein Praktikum, welches einige Aspekte des ASE vermittelt. Die inhaltlichen Anforderungen benötigen ein entsprechendes Vorwissen, so dass in erster Linie Bachelor-Studierende im letzten Studienjahr als Teilnehmer in Frage kommen.

Beschreibung des Moduls

Im folgenden Abschnitt wird das Praktikum mit der begleitenden Vorlesung vorgestellt und die zu vermittelnden Kompetenzen abgeleitet.

Teilnehmer

Das Praktikum ist Bestandteil des Moduls *Embedded Systems und Software Engineering* für Studierende im sechssemestrigen Studiengangs Mechatronik an der Hochschule Osnabrück. Das Modul im fünften Fachsemester besteht aus einer Vorlesung (2 SWS) und einem begleitenden Praktikum (2 SWS) und entspricht 5 Leistungspunkten nach ECTS.

Der Mechatronik-Studiengang an der Hochschule Osnabrück ist nicht anwendungsspezifisch und in der Ausbildung breit angelegt. Aufgrund der eingangs beschriebenen Bedeutung der Automobilindustrie in der Region Osnabrück werden Vorgehensweisen, Werkzeuge und Methoden praxisnah vermittelt.

Lernziele

Aufgrund des Zeitrahmens verzichtet das Praktikum auf den vollständigen Durchlauf aller Phasen im Software-Entwicklungsprozess. Stattdessen werden einzelne Schwerpunkte gesetzt, die sich aus dem spezifischen automobilen Vorgehen ergeben. Die nachfolgende Definition von Lernzielen anhand von Kompetenzfeldern orientiert sich an der Darstellung von (Böttcher und Thurner, 2011).

Sachkompetenz

Die Studierenden lernen das Vorgehensmodell in der Automobilindustrie kennen. Hierbei werden aufgrund der arbeitsteiligen Fahrzeugentwicklung die abgeleiteten Konsequenzen verdeutlicht. Neben einer integrierten Entwicklungsumgebung für das eingebettete System (µVision, Fa. Keil) wird ein Werkzeug zur Modellierung verteilter Systeme von Steuergeräten vermittelt. Dieses Werkzeug (CANoe, Fa. Vector) kann in unterschiedlichen Phasen (u.a. im Entwurf, Simulation, Test) des Entwicklungsprozesses eingesetzt werden.

Methodenkompetenz

Das Praktikum soll insbesondere das *Denken in Schnittstellen* als Methode zur Abstraktion und zur Modularisierung fördern. Die Betrachtung der einzelnen Steuergeräte als *Black Box* macht die Funktionalität an einer Schnittstelle sichtbar und vermeidet den Blick auf interne Details oder Abläufe in frühen Phasen der Entwicklung. Dieser Ansatz reduziert die Komplexität und lässt dadurch große Software-Systeme handhabbar werden.

Als zweiten Ansatz zur Reduzierung der Komplexität soll das *Denken in Zustandsmaschinen,* das Beschreiben von Verhalten, vertieft werden. Während Schnittstellen im Allgemeinen nur eine statische Sicht auf eine Komponente oder System beschreiben, ist für ein vollständiges Verständnis die dynamische Sicht entscheidend. Zustandsdiagramme ggfs. in Kombination mit Sequenzdiagrammen sind durch die Studierenden hierbei einzusetzen.

Sozialkompetenz

Die arbeitsteilige Entwicklung wird im Praktikum nachgestellt. Hierzu entwickeln mehrere Teams unterschiedliche Steuergeräte, die gemeinsam in ein Fahrzeug integriert werden. Da die Abnahme des Praktikums hierbei nur als Ganzes erfolgt, sind die Studierenden aufgefordert, über die eigenen Teamgrenzen hinweg offene Punkte anzusprechen und Lösungen gemeinsam zu erarbeiteten. Hierbei bilden die Schnittstellen üblicherweise die Reibungspunkte in der Zusammenarbeit. Zudem müssen die Teams sich zeitlich koordinieren, um gemeinsame Tests und Erprobungen durchzuführen.

Vorlesung

Nicht alle der vorgenannten Grundlagen können in diesem Bachelor-Modul vorausgesetzt werden und sind daher Bestandteil des ersten Teils der begleitenden Vorlesung. Die Schwerpunkte sind neben dem grundsätzlichen Aufbau von Echtzeitsystemen hierbei: Hardware-nahe Software-Entwicklung und Betriebssysteme (Scheduling-Verfahren sowie Zugriff auf gemeinsame Ressourcen).

Um die Studierenden auf die Bedeutung der Automobilindustrie in Deutschland hinzuführen, beginnt der zweite Teil mit einem Exkurs zum Thema Technologien *Made in Germany*. Hierbei werden exemplarisch die Errungenschaften und Erfindungen von Werner von Siemens (Dynamoelektrisches Prinzip, 1867), Robert Bosch (Zündkerze, 1902) und Konrad Zuse (programmgesteuerter Rechner, 1941) vorgestellt. Hieran schließt sich eine Frage zur Aktivierung der Studierenden an: *Was sind aus Ihrer Sicht die wichtigsten Erfindungen aus Deutschland der letzten 30 Jahre?*

Nur wenige der Studierenden haben hierzu eine eigene Vorstellung. Offensichtlich sind die meisten Technologien wie Suchmaschinen, soziale Netzwerke oder moderne Mobiltelefone von amerikanischen Unternehmen erfunden und entwickelt worden. Nur wenigen Studierenden fällt spontan das

in Deutschland entwickelte MP3-Format (Karlheinz Brandenburg, 1987) ein. Dass das *Controller Area Network* oder abgekürzt CAN (Siegfried Dais und Uwe Kiencke, 1985) eine deutsche Technologie ist, die inzwischen in jedem Fahrzeug weltweit eingesetzt wird, ist den Studierenden unbekannt.

In den nachfolgenden Lehreinheiten werden die CAN-Bus Grundlagen vermittelt und somit die Voraussetzungen für das Praktikum geschaffen.

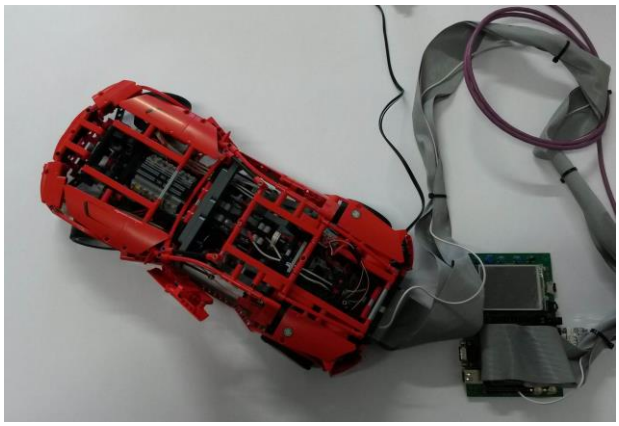


Abb. 3: Erweitertes Modellfahrzeug mit CAN

Praktikum und Aufgabenstellung

Eine vollständige Funktionsentwicklung eines Steuergerätes ist aufgrund der zur Verfügung stehenden Zeit und der erforderlichen Voraussetzungen nicht Schwerpunkt des Moduls. Stattdessen wird ein einfaches Beleuchtungssystem (Blinker, Fahrlicht, etc.) als anschauliches Beispiel eingesetzt. Dieses bietet den Vorteil, dass die Anforderungen einfach darzustellen sind, da viele der Studierenden einen Führerschein und ggfs. ein eigenes Fahrzeug besitzen.

Abbildung 3 zeigt das im Praktikum eingesetzte Modellfahrzeug (Lego Super Car 8070). Dieses um eine Lichtsteuerung sowie eine Motoransteuerung für die Fahrertür erweitert worden. Der dargestellte Mikrocontroller empfängt über den CAN-Bus Nachrichten zur Ansteuerung sowie versendet periodisch Sensordaten (Helligkeit).

Insgesamt sind 6 Steuergeräte in einer vereinfachten Ausbaustufe in einem Fahrzeug zu integrieren:

- Bedieninstrument
- Getriebe
- ABS (Bremse)
- Zentralverriegelung
- Zündschloss
- Licht

Die zuletzt aufgeführten ECU werden als Simulation bzw. als eingebettetes System für das Modellfahrzeug vorgegeben. In Abbildung 4 ist der voll-

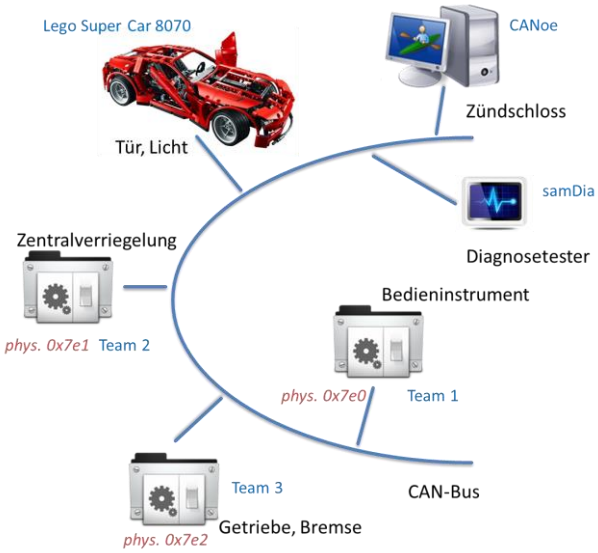


Abb. 4: Arbeitsteilige Entwicklung im Praktikum

ständige Praktikumsaufbau skizziert. An einem gemeinsamen CAN-Bus ist das Lego Super Car 8070, die CANoe-Simulation für das Zündschloss und 4 weitere Steuergeräte angeschlossen, die das Bedieninstrument, die Zentralverriegelung und das Getriebe sowie die Bremse realisieren.

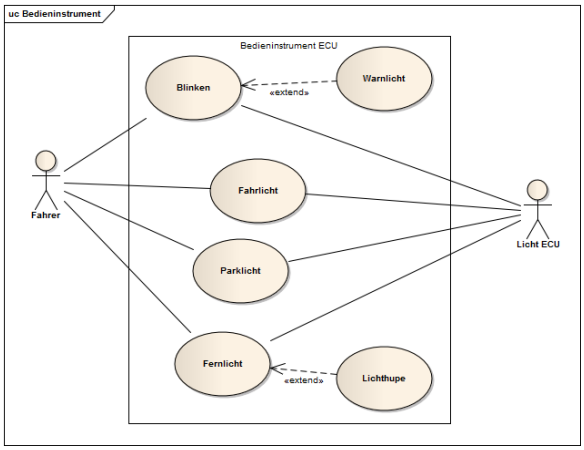


Abb. 5: Anwendungsfälle für Bedieninstrument

Die Phase der Anforderungsanalyse im Entwicklungsprozess wird durch Vorgabe von Anwendungsfällen (siehe Abbildung 5) und textuellen Beschreibungen verkürzt. Nichtsdestotrotz ergeben sich Spielräume in der Ausgestaltung der Lichtfunktionen (beispielsweise Abschaltung des Abblendlichtes während Fernlicht).

Aus den vorgegebenen Anforderungen haben die einzelnen Teams das entsprechende Steuergerät zu entwickeln. Die Realisierung erfolgt in vier Phasen entsprechend dem in der Automobilindustrie angewendeten Vorgehen:

1. Vollständige Simulation: die grundsätzliche Funktionsweise des zu entwickelnden

Steuergeräts wird überprüft und die Interaktion mit dem Lego Super Car 8070 getestet (siehe Abbildung 6).

2. Reales Steuergerät mit simulierter Umgebung: die Simulation des zu entwickelnden Steuergeräts wird durch eine Implementierung auf einem eingebetteten System ersetzt.
3. Reale Steuergeräte (ohne Simulation): die simulierte Umgebung wird durch das Fahrzeug (Steuergerät für Beleuchtung und Tür) sowie durch weitere Steuergeräte ersetzt. Die Zündung wird weiterhin über CANoe simuliert.
4. Diagnose: Die Funktionalität des Steuergerätes wird um Möglichkeiten zur Diagnose erweitert. Hierbei wird in Abhängigkeit des Steuergerätes auf Anfragen des Diagnosetesters geantwortet.

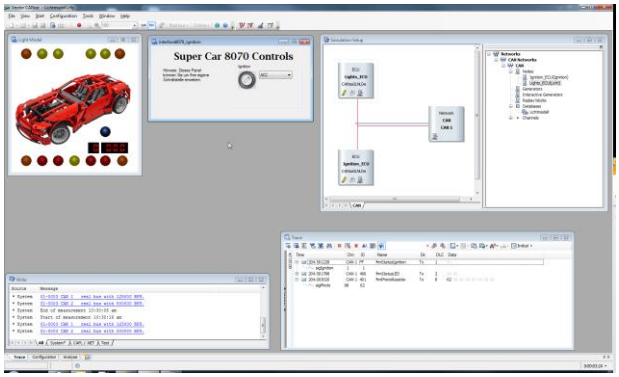


Abb. 6: Modellierung und Simulation mit CANoe

Durch die detaillierte Vorgabe der Anforderungen können die Studierenden eine manuelle Verifikation in den unterschiedlichen Phasen ihrer Software-Entwicklung durchführen. Zusätzlich werden simulierte Komponenten bereitgestellt, so dass Tests teilautomatisiert werden können.

Erfahrungen und Empfehlungen

Die folgenden Abschnitte fassen die Beobachtungen und Erfahrungen zusammen. Hieraus lassen sich Empfehlungen für ähnliche Praktika ableiten.

Konzentration auf wenige Aspekte

Ein häufiger Ansatz in der Lehre als auch in der Literatur ist die Vorstellung von verschiedenen Entwicklungsprozessen, vom Wasserfallmodell, iterative Ansätzen, V-Modell oder agilen Ansätzen. Hierbei werden die einzelnen Phasen vertieft. Oftmals werden zusätzlich bestimmte Methoden, beispielsweise zur Modellierung in UML von Software, miteingeführt.

Aus den Lernzielen abgeleitet beschränkt sich die durchgeführten Praktikumsversuche auf der Vorstellung sowie der Vertiefung nachfolgender Aspekte:

- Die Beschreibung von Verhalten durch Zustandsmaschinen; hierbei entstehen auch verteilte Zustandsmaschinen, da der Systemzustand nicht zentral sondern verteilt in den einzelnen Netzknoten vorliegt.
- Durch das Zusammenspiel von jeweils drei Teams ergibt sich eine arbeitsteilige Entwicklung der Steuergeräte.
- Entwurf von Schnittstellen und die Umsetzung von Nachrichten für den CAN-Bus.
- Entwurfsprozess: vollständige Simulation der Netzknoten, Kombination aus einer Zielhardware und der Restbussimulation und gemeinsame Integration aller Steuergeräte.

Die Fokusbestimmung macht einen Auswahlprozess notwendig, der auch die Voraussetzungen der Studierenden berücksichtigt. So kann beispielsweise bei Informatikern der Schwerpunkt eher auf der Modellierung von Softwarekomponenten, bei Ingenieuren ggfs. eher im Bereich der Steuerungs- und Reglungstechnik liegen.

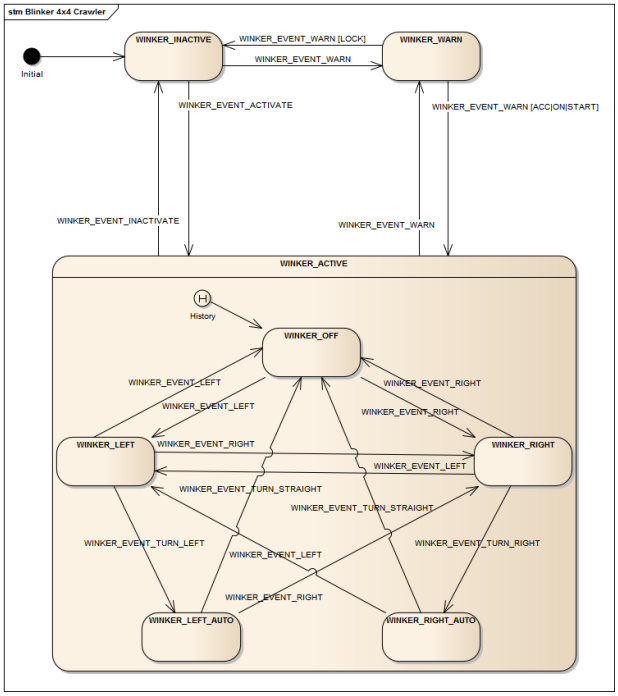


Abb. 7: Zustandsmaschine für Blinker

Abbildung 7 gibt beispielhaft die Zustandsmaschine für den Blinker mit einer automatischen Abschaltung wieder. Die Studierenden erlernen aus einer textuellen Anforderungsbeschreibung das Verhalten zu modellieren. Ausgehend von einfachen Zustandsmaschinen werden weiterführende Methoden (Hierarchie und Historie) vermittelt, so

dass komplexe Verhaltensmuster angemessen dargestellt werden können.

Dosierter Einsatz von Werkzeugen

Besonders in der modellbasierten Entwicklung im automotiven Umfeld existiert eine Reihe von sehr spezialisierten Werkzeugen. Ohne den Werkzeugeinsatz ist beispielsweise eine Software Entwicklung für AUTOSAR-kompatible Komponenten nicht denkbar.

Die folgenden Nachteile ergeben sich hieraus:

- Die Werkzeuge bieten im Allgemeinen keinen Lernmodus o.ä. an und orientieren sich an den konkreten Aufgaben in der beruflichen Praxis. Ein pädagogischer Einstieg ist nicht möglich.
- Die graphische Bedienschnittstelle ist stark technisch ausgerichtet und orientiert sich an den Belangen der Industrie (*Expertenwissen*).
- *A fool with a tool is still a fool*: Aufgrund der Abstraktion, die Werkzeuge bei der Entwicklung einführen, verlieren Studierende Transparenz in ihrem Vorgehen. Konsequenzen bleiben unklar. Bei Fehlern *funktioniert etwas nicht* und eine Ursachensuche findet dann nicht statt.
- Ein Zwiespalt in der Ausbildung: zum einen sollen Studierende Methoden und Kompetenzen erlernen und nicht zum *Werkzeugbediener* ausgebildet werden. Zum anderen soll praxisnah und berufsbefähigend ausgebildet werden. Hierzu gehören so dann auch Kenntnisse solcher industriellen Werkzeuge.

Schon der zeitliche Aufwand steht einem verstärkten Einsatz von Werkzeugen im Wege. Viele Werkzeughersteller bieten 3-5 tägige Schulungen zu Ihren Produkten an. Diese entspricht ungefähr dem Umfang einer vollständigen 2 SWS Veranstaltung.

Abb. 8: Physisches Lichtmodell auf Basis Lego

In dem vorliegen Modul werden neben einer integrierten Entwicklungsumgebung (im Wesentlichen nur zum Editieren und zum Übersetzen) ein weiteres Werkzeug CANoe eingesetzt, welches es ermöglicht, einen typischen Entwicklungszyklus im Automotive Software Engineering zu durchlaufen.

Verwendung von physischen Modellen

Ein immer wieder nicht zu unterschätzender Faktor ist die Benutzung von physischen Modellen in der Ausbildung von Studierenden. Dieser positive Effekt auf die Motivation der Studierenden wird auch anderweitig beobachtet (Uelschen und Eikerling, 2011). In dem Praktikum werden keine realen Steuergeräte oder Fahrzeugkomponenten entworfen, sondern dieses wird durch ein Lego-Technik Modell (siehe Abbildung 8) in sehr einfacher Weise ersetzt.

Obwohl die Entwicklung und der Test auch vollständig auf dem Desktop-Rechner in einer Simulation erfolgen können, hat der Einsatz von haptischen Modellen Vorteile:

- Die Studierenden werden aus ihrer *Lebenswelt* abgeholt. Sie haben die Modelle in Ihrer Jugendzeit in einer anderen Umgebung kennengelernt.
- Reale Modelle sind weniger abstrakt als simulierte Anzeigen auf dem Bildschirm und verdeutlichen den Studierenden einen Einsatz in der Praxis.

Das in (Nazareth und Wurm, 2013) verwendete Modellfahrzeug im Maßstab 1:18 wird als Plattform zur modellbasierten Entwicklung eingesetzt.

Im Wintersemester 2014/15 ist ein weiteres Fahrzeugmodell im Praktikum eingesetzt worden (siehe Abbildung 9), welches teilweise eine alternative Funktionsweise zur Beleuchtung realisiert (beispielsweise automatische Rückstellung der Blinker bei Geradeausfahrt). Zudem ist die Ansteuerung der LEDs platzsparend auf der Basis Arduino mit CAN-Shield realisiert.

Abb. 9: Alternativ eingesetztes Lichtmodell

Fazit

Der vorliegende Beitrag zeigt, wie spezifische Aspekte des Automotive Software Engineering in einem Bachelor-Modul integriert werden können. Der Schwerpunkt liegt hierbei auf der arbeitsteiligen Entwicklung, den Entwurf von Schnittstellen und der dynamischen Modellierung mit Zustandsmaschinen. Bei der Entwicklung von Steuergeräten zu Ausbildungszwecken kann im Allgemeinen nicht auf reale Komponenten der Automobilindustrie zurückgegriffen werden, da die Schnittstellen nicht öffentlich und notwendige Informationen beim OEM nicht erhältlich sind. Stattdessen muss auf einfache Modelle zurückgegriffen werden.

Das Praktikum ist in der vorgestellten Weise inzwischen dreimal erfolgreich durchgeführt worden. Zukünftig soll die Verifikationsphase vertieft werden, da sich herausgestellt hat, dass ein systematischer Test bei den Studierenden insbesondere in den ersten Phasen verkürzt wird.

Das Praktikum bietet zudem zahlreiche Möglichkeiten zur Erweiterungen. Zur Vertiefung von Zustandsmaschinen kann beispielsweise das Netzmanagement in die Aufgabenstellung integriert werden. Die Anforderungsanalyse ist durch die Vorgabe von konkreten Anwendungsfällen im aktuellen Praktikum stark verkürzt. Alternativ besteht die Möglichkeit, die Anforderungen aus der Straßenverkehrsordnung durch die Studierenden ableiten zu lassen.

Literatur

Böttcher, A., Thurner, V. (2011): Kompetenzorientierte Lehre im Software Engineering. In: Ludewig, J., Böttcher, A. (Hrsg.), Software Engineering im Unterricht der Hochschulen (SEUH), München, CEUR Workshop Proceedings Vol. 695, http://ceur-ws.org/Vol-695/, S. 33-39.

Broy, M. (2006): Challenges in Automotive Software Engineering. In: Proceedings of the 28th International Conference on Software Engineering (ICSE '06), ACM, S. 33-42.

Chrissis, M., Konrad, M., Shrum, S. (2011): CMMI for Development: Guidelines for Process Integration and Product Improvement, 3rd edition, Addison-Wesley.

Fortiss (2011): Mehr Software (im Wagen). Abschlussbericht des vom Bundesministerium für Wirtschaft und Technologie geförderten Verbundvorhabens eCar-IKT-Systemarchitektur für Elektromobilität.

Gebhardt, V., Rieger G., Mottok, J., Gießelbach C. (2013): Funktionale Sicherheit nach ISO 26262: Ein Praxisleitfaden zur Umsetzung, dpunkt.

Höhn, H., Sechser B., Dussa-Zieger K., Messnarz, R., Hindel B. (2009): Software Engineering nach Automotive SPICE, dpunkt.

Lucke, H., Schaper, D., Siepen, P., Uelschen, M., Wollborn, M. (2007): The Innovation Cycle Dilemma. In: Koschke, R., Herzog, O., Rödiger, K.-H., Ronthaler, M. (Hrsg.), INFORMATIK 2007, Band 2, S. 526-530.

Mössinger, J. (2010): Software in Automotive Systems. In: Software, IEEE, S. 92-94.

Nazareth, D., Wurm, C. (2013): Modellbasierte Entwicklung einer Lichtsteuerung für ein Rapid Prototyping System. In: Halang, W. (Hrsg.), Kommunikation unter Echtzeitbedingungen, Springer, S. 49-58.

Pfleging, B., Schneegass S., Kern, D., Schmidt, A. (2014): Vom Transportmittel zum rollenden Computer – Interaktion im Auto. In: Informatik Spektrum, GI, S. 418-422.

Pretschner, A., Broy, M., Krüger, I., Stauner, T. (2007): Software Engineering for Automotive Systems: A Roadmap. In: Future of Software Engineering 2007, IEEE, S. 55-71.

Ritter, S. (2004): Software auf der Straße: Herausforderungen des Software-Engineering in der Automobilindustrie. In: Objektspektrum, 4, S. 59-62.

Schäuffele, J., Zurawka, T. (2012): Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen, 5. Auflage, Springer.

Sommerville, I., (2012): Software Engineering, 9. Auflage, Pearson.

Uelschen, M., Eikerling, H.-J. (2011): An Introductory Course on Software Engineering on Self-Organization in Swarm Robotics. In: Proceedings of the 24th Conference on Software Engineering Education and Training (CSEE&T), IEEE, S. 333-342.

Weinmann, U. (2003): Software im Automobil – Anforderungen und Chancen. In: Siedersleben, J., Weber-Wulff, D. (Hrsg.), Software Engineering im Unterricht der Hochschulen (SEUH), dpunkt Verlag, Heidelberg, S. 1-7.

Winner, H., Hakuli, S., Wolf, G. (2012) (Hrgs.): Handbuch Fahrerassistenzsysteme , 2. Auflage Springer.

Zimmermann W., Schmidgall R. (2014): Bussysteme in der Fahrzeugtechnik, 5. Auflage, Springer.

Projekte in der Wirtschaft

Juliane Siegeris und Jörn Freiheit

Hochschule für Technik und Wirtschaft Berlin
Informatik und Wirtschaft
siegeris@htw-berlin.de | freiheit@htw-berlin.de

Zusammenfassung

Der Beitrag stellt die Projektveranstaltungen des Studiengangs Informatik und Wirtschaft (für Frauen) an der HTW Berlin vor. Das Besondere bei diesen Projekten ist, dass die Studentinnen an realen Aufgabenstellungen arbeiten, die von Unternehmen gestellt und begleitet werden. Für jedes Wintersemester werden dafür zwischen 10-12 verschiedene Partner aus der Region gewonnen. Diese begleiten ein Team von etwa 5-7 Studentinnen ein Semester lang. Diese Art realer Projekte steht nicht am Ende des Studiums, sondern fordern die Studentinnen das erste Mal im dritten und dann ein weiteres Mal im fünften Semester ihres Bachelor-Studiums. Der Beitrag stellt Rahmenbedingungen und Durchführung der Projekte vor und diskutiert Vor- und Nachteile der Veranstaltung im Kontext des mono-edukativen Studienangebots.

Einleitung

Es ist inzwischen an vielen Universitäten und Hochschulen üblich, dass die Studierenden selbständig Projekte bearbeiten und damit die in den fachlichen Vorlesungen erworbenen Fähigkeiten erproben und festigen, vgl. z.B. (Kleuker u. Thiesing, 2011; Lewerentz u. Rust, 2001; Liebehenschel, 2013). Ein weiteres Ziel dieser Projekte ist es, typische Vorgehensweisen selbst anzuwenden und das Arbeiten im Team zu erlernen. Mehr als in jeder Frontalveranstaltung eigenen sich die Projekte dazu, neben den technischen Fähigkeiten auch wertvolle Softskills zu trainieren.

Dass solche Fähigkeiten mehr denn je gefragt sind, lässt sich der Literatur, vgl. (Rolf u. a., 2013; Vigenschow u. a., 2011), aber auch einer Studie, vgl. (SIBB e.V., 2012) entnehmen. Bei dieser Umfrage, die durch den Verband der IT- und Internetwirtschaft in Berlin und Brandenburg (SIBB) unter 104 Unternehmen aus der ICT-Branche im Jahr 2012 durchgeführt wurde, gaben 66% der Befragten auf die Frage „in welcher Hinsicht die Unternehmen die Hochschulausbildung für verbesserungswürdig halten“ an, dass „mehr Projekte unter Realbedingungen“ durchgeführt werden sollten. Dies war somit der höchste Verbesserungsbedarf der Industrie an die Hochschulausbildung, noch vor bspw. „konzeptionellen Fähigkeiten“ (46%) und „modernen Technologien im Rahmen der Fachinhalte“ (45% - Mehrfachnennungen waren möglich). Siehe dazu auch Abbildung 1.

Abbildung 1: Wünsche der Unternehmen an Ausbildungsinhalte. Quelle: (SIBB e.V., 2012)

Um die Studierenden neben dem Fachwissen mit den im späteren Berufsfeld notwendigen Schlüsselkompetenzen auszustatten, wurde bei der Neukonzeption des Studiengangs Informatik und Wirtschaft (für Frauen) ein besonderes Augenmerk auf die Entwicklung von Methoden und Sozialkompetenz gelegt. Im Curriculum wird dies durch eine größere Anzahl von Lehrveranstaltungen deutlich, in denen die Vermittlung von Softskills im Vordergrund steht. Ein weiterer Schwerpunkt wird auf die Anwendung von didaktischen Methoden gelegt, die lebendiges Lernen unterstützen. Bei diesen Methoden steht die Selbsterschließung von Bildungsinhalten durch die Studierenden im Vordergrund. Die Studierenden werden zu aktiven Lernern und die Lehrenden werden zum Lern-Coach. Beispiele sind Lernteamcoaching, Hackathon oder studentische Projekte, vgl. (Siegeris u. Krefting, 2014). In diesem Papier sollen zwei Lehrveranstaltungen die „Projekte in der Wirtschaft“ (im 3. Fachsemester) und die Lehrveranstaltung „Projekt“ (im 5. Fachsemester) näher betrachtet werden. Ein Merkmal dieser Veranstaltungen ist die konsequente Einbindung von realen Auftraggebern. Dies ist unserer Meinung nach eine Besonderheit auch im Vergleich zu Projektveranstaltungen anderer Studiengänge der HTW oder anderer Hochschulen, vgl. z.B. (Liebehenschel, 2013). Während häufig ausgedachte Szenarien in abgewandelter Form jedes Jahr wiederverwendet und teilweise von mehreren Teams parallel bearbei-

tet werden, werden bei uns jedes Jahr 10-12 unterschiedliche Projekte mit wechselnden Auftraggebern organisiert.

Projekte in der Wirtschaft - Verortung

Die Lehrveranstaltungen „Projekte in der Wirtschaft“ und „Projekt“ gehören zum Curriculum des Studiengangs „Informatik und Wirtschaft (für Frauen)“ der HTW. Dieser Studiengang ist noch jung – die ersten Studierenden wurden 2009 immatrikuliert. Als Reaktion auf den Fachkräftemangel und die Unterrepräsentation von Frauen in der IT-Branche wurde er als Frauenstudiengang konzipiert.

In dem Bachelorstudiengang wird den Studentinnen durch eine Mischung aus typischen Informatikveranstaltungen (insgesamt etwa 60 Prozent), grundlegenden Wirtschaftsfächern (etwa 10 Prozent) und Softskills / Englisch (etwa 10 Prozent) ein für heutige Ansprüche des Arbeitsmarktes passendes Profil gegeben. Eine Besonderheit ist der hohe Praxisanteil (verbleibende 20 Prozent!) der durch explizite Lehrveranstaltungen (Projekte in der Wirtschaft, Exkursionen, Praktikum und Projekt) erreicht wird. Mit dem hohen Praxisanteil sollen mehrere Ziele erreicht werden: fachliche Kompetenzen werden selbständig erarbeitet bzw. angewendet und überfachliche Kompetenzen werden trainiert. Ganz nebenbei wird den Studierenden ein möglichst realistisches Bild des zukünftigen Arbeitsplatzes bzw. werden reale Kontakte dorthin vermittelt.

Die Veranstaltung „Projekte in der Wirtschaft“ ist (schon) im 3. Semester des Bachelor-Studiums angesiedelt. Bis dahin haben die Studentinnen Grundlagen der Informatikausbildung: Programmierung I und II (Java), Softwareengineering, Rechnerarchitektur und Betriebssysteme, Grundlegende Konzepte der Informatik sowie einführende Veranstaltungen der Wirtschaft (BWL, Rechnungswesen) und Mathematik gehört. Im 3. Semester belegen die Studentinnen u.a. parallel Projektmanagement, Webtechnologien und Datenbanken. Die Veranstaltung „Projekt“ im 5. Fachsemester ist ähnlich organisiert wie die Veranstaltung im 3. Semester. Hier bringen die Studentinnen auch ihr Wissen aus ihrem ersten Projekt und dem Praktikum ein.

Im Rahmen der Projektveranstaltungen erhalten die Studentinnen die Gelegenheit, reale Projekte selbstständig durchzuführen. Das Besondere: echte Unternehmen der Region treten dabei als Auftraggeber auf. Die Unternehmen formulieren die Aufgabenstellung und begleiten ein Team von 5-7 Studentinnen bei der Bearbeitung durch das ganze Semester.

Die Aufgabe der Professor_innen besteht in der Akquise der Auftraggeber - immerhin etwa 10-12 verschiedene Unternehmen pro Semester, der Unterstützung des Teambuilding- und des Projektzuordnungs-Verfahrens am Anfang, sowie in der eher formalen Begleitung durch das Semester: regelmäßige Projekleiterinnen-Sitzungen, Teilnahme an ausgewählten Teamtreffen, Organisation von Zwischen- und Abschlusspräsentation und natürlich der Bewertung.

Ablauf der Lehrveranstaltung – von der Akquise bis zur Bewertung

Im Folgenden werden die verschiedenen Stationen eines Projekts, von der Anbahnung bis zu Abschluss und Bewertung, vorgestellt:

Akquise der Unternehmen

Die Unternehmen werden mit Hilfe eines Handzettels, der die Rahmenbedingungen, den Aufwand und die Vorteile für die Auftraggeber skizziert, angesprochen. Sind sie interessiert, beschreiben sie ihr Projekt in Form eines Projektsteckbriefs. In dem ca. 2-seitigen Dokument beschreiben die Auftraggeber die Ausgangssituation, sowie Projektziel und Projektanforderungen. Bei der Beschreibung des Projektziels legt der Auftraggeber fest, was die notwendigen und darüber hinausgehenden wünschenswerten Projektergebnisse sind, die für einen erfolgreichen Projektabschluss erreicht werden sollen.

Die Steckbriefe werden den Studentinnen in der begleitenden Moodle-Umgebung zur Verfügung gestellt.

Vorstellung der Lehrveranstaltung

Im ersten Termin der Lehrveranstaltung werden die Studentinnen auf das Projekt vorbereitet. Der generelle Ablauf und die Bewertungskriterien werden vorgestellt. Zudem wird auf die eventuell konfliktreiche Arbeit im Team hingewiesen. Die Dozenten erläutern Prinzipien im Umgang und in der Kommunikation mit den Auftraggebern.

In diesem Termin erfolgt auch die Zusammenstellung der Teams. Dabei werden im Losverfahren Teams gelost, die in möglichst gleicher Anzahl aus Studentinnen des 3. und des 5. Semesters bestehen. Außerdem werden allen Teams alle Projektsteckbriefe zur Verfügung gestellt. Es ist jedoch zu diesem Zeitpunkt noch nicht klar, welches Team welches Projekt durchführen wird. Dies wird erst nach den Auftraggeberpräsentationen ermittelt.

Auftraggeberpräsentationen

Die Auftraggeberpräsentationen finden spätestens in der zweiten Woche des Semesters statt. Dazu kommen die Ansprechpartner der Unternehmen an die Hochschule und präsentieren ihre Projekte in jeweils 20-minütigen Vorträgen. Die Teams stellen jeweils Fragen zu den Projekten. Diese beinhalten sowohl fachliche Aspekte als auch organisatorische. So wird bspw. häufig gefragt, ob der Auftraggeber die Präsenz des Teams im Unternehmen wünscht und wenn ja, wie oft, wie der Auftraggeber sich die Abstimmung mit dem Projektteam vorstellt und wie viel

Zeit der Ansprechpartner zur fachlichen Unterstützung vorsieht. Außerdem werden eventuelle Unterschiede zwischen der Projektbeschreibung im Vortrag und derer im Projektsteckbrief geklärt. Die Projektteams (die zu diesem Zeitpunkt noch nicht wissen, welches Projekt sie bearbeiten werden) interessieren sich für alle Projekte gleichermaßen und erstellen während der Präsentationen eine Prioritätenliste der Projekte, die sie am liebsten bearbeiten würden.

Zuordnung der Projekte

Die Zuordnung der Projekte findet wenige Stunden nach den Auftraggberpräsentationen, spätestens jedoch am folgenden Tag statt. Die Zuordnung wird durch die Dozenten moderiert. Die Grundidee für die Zuordnung der Projekte besteht darin zu versuchen, dass die Teams sich die Projekte selbst aussuchen, um eine möglichst große Motivation für die Projektbearbeitung zu gewährleisten.

In einer geheimen Wahl schreiben die Teams ihr Wunschprojekt auf einen Zettel. Alle Projektnamen werden nebeneinander an die Tafel geschrieben. Unter die Projektnamen werden nach der ersten Wahlrunde nun die Teams notiert, die sich für das jeweilige Projekt entschieden haben. Die Projekte, für die sich jeweils genau ein Team entschieden hat, sind somit zugeordnet. Die Projekte, für die sich keines oder mehrere Teams entschieden haben, stehen in einer zweiten Runde erneut zur Wahl.

An dieser zweiten Runde beteiligen sich also nur noch die Teams, die sich zusammen mit einem oder mehreren anderen Teams für ein gleiches Projekt entschieden hatten. Sowohl die zweite als auch die dritte und letzte Wahlrunde laufen nach dem gleichen Prinzip ab wie die erste. Zwischen den Runden wird den Studentinnen etwas Zeit für Abstimmungen im Team oder mit anderen Teams eingeräumt.

Nach der dritten Runde droht das Losverfahren. Darin würden alle noch verfügbaren Teams den noch verfügbaren Projekten zugelost werden. Häufig entsteht jedoch nach der zweiten Runde die Situation, dass sich um vielleicht drei Projekte jeweils genau zwei Teams streiten, während drei weitere Projekte bisher nicht gewählt wurden. Bei dem Losverfahren nach der dritten Runde hätten diese sechs Teams jedoch dann nur eine 1/6tel Chance, ihr Lieblingsprojekt zu bekommen, während eine interne Wahl zwischen jeweils zwei Teams die Chancen auf 50% sichert. Natürlich versuchen die Teams zu vermeiden, ein gänzlich ungewolltes Projekt zu bekommen. Deshalb finden nach der zweiten Wahlrunde häufig „interne“ Losrunden statt. Dieses Verfahren hat im laufenden Durchgang sehr gut geklappt. Nach der dritten Runde ging ein Raunen durch die Gruppe, da alle Projekte verteilt wurden, ohne das abschließend gelost werden musste.

Start der Projekte

In einer Veranstaltung, die möglichst zeitnah nach den Projektzuordnungen erfolgt, wird das „Teambuilding“ gefördert. Dabei soll sowohl das eigene Rollenverhalten reflektiert werden, als auch eine Einordnung in der Gruppe gefunden werden. Angeregt durch (Kleuker u. Thiesing, 2011) verwenden wir dafür einen Belbin-Test (Belbin, 2012) den jede Studentin für sich absolviert. Die Ergebnisse des Tests werden dann in der Gruppe diskutiert und nach dem anschließenden Teamspiel nochmal reflektiert. Ziel ist es die Rollenverteilung im Team zu unterstützen und im besten Fall schon eine Projektleiterin zu bestimmen. Die anonymisierten Ergebnisse der Belbin-Tests wurden von den Professor_innen eingesammelt. Ziel ist es die gefundenen Rollenprofile in der reinen Frauengruppe mit Profilen aus ähnlichen aber gemischten Projektveranstaltungen anderer Hochschulen (Kleuker u. Thiesing, 2011) zu vergleichen.

Für ein erstes Teambuilding, sowie als Basis für die anschließende Rollen-Reflektion wird anschließend ein spielerisches Miniprojekt durchgeführt. Dabei geht es z.B. darum, einen möglichst hohen Turm aus einer begrenzten Anzahl aus Spaghettis, Marshmallows und Büroklammern zu bauen. Durch die ungewöhnliche Aufgabe entsteht eine sehr ungezwungene Atmosphäre, die es den Studierenden erleichtert miteinander ins Gespräch und ins erste „produktive“ Arbeiten zu kommen und anschließend auch das Teamverhalten zu diskutieren und zu analysieren.

Zur Vorbereitung auf die eventuell konfliktbehaftete Zusammenarbeit im Team werden die Studentinnen gebeten, erst einzeln und dann in ihren Gruppen Regeln zu erarbeiten, die für die Zusammenarbeit im Team gelten sollen. Diese Teamregeln werden auf einem gemeinsamen Plakat festgehalten und am Ende der Lehrveranstaltung von allen Teammitgliedern unterschrieben. Das Plakat wird abfotografiert und als Teamvertrag in die begleitende Moodleumgebung hochgeladen.

Durchführung der Projekte

Als erste inhaltliche Aufgabe werden die Teams aufgefordert, selbstständig das Kick-off mit ihrem jeweiligen Auftraggeber zu organisieren und zu gestalten. Unterstützend greift hierbei die Lehrveranstaltung „Projektmanagement“ ein. Diese Veranstaltung ist das ganze dritte Semester über auf die Begleitung der Projekte ausgelegt. Die Proketmanagement-Dozentin steht den Studentinnen durchweg als Ansprechpartnerin für die organisatorischen Aspekte der Projekte zur Seite.

Die 10-12 Projektteams werden fachlich von allen drei Dozent_innen des Studiengangs unterstützt. Jedes Team ist dabei einer Professorin bzw. einem Professor zugeordnet, welche(r) sich regelmäßig (meis-

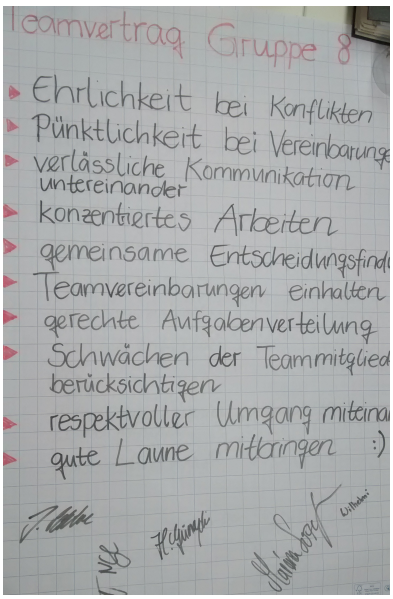


Abbildung 2: Beispiel eines Plakates mit Teamregeln

tens 14-tägig) mit den Projektleiterinnen und zwischendurch mit einzelnen Teams trifft, um den Projektfortschritt zu besprechen und gegebenenfalls auch fachlich unterstützend einzugreifen. In Vorbereitung dieser Sitzungen erstellen die Teams Controllingberichte. In diesen Berichten wird nicht nur über den aktuellen Status des Projektes, die Ergebnisse der letzten, die geplanten Projektschritte für die nächsten 14 Tage, über erkannte Risiken und Handlungsbedarfe berichtet, sondern jede Studentin gibt einen eigenen Arbeitsnachweis über die im Projekt geleisteten Arbeitsstunden und die erzielten Ergebnisse ab. Dieser Arbeitsnachweis dient auch der Diskussion im Team über eine Gleichverteilung der Arbeitsbelastung.

Die Teams präsentieren in regelmäßigen Treffen (mindestens 2-wöchig, häufig wöchentlich) die Projektfortschritte den Auftraggebern und stimmen Erreichtes und Geplantes mit ihnen ab. Diese Treffen finden meistens bei den Auftraggebern statt. Die Vorlesungspläne sowohl des dritten Semesters als auch des fünften Semesters sind so angelegt, dass beide einen gemeinsamen Tag nur für die Projekte zur Verfügung haben.

Zwischen- und Endpräsentation

Anfang Dezember, also zwei Monate nach Beginn der Projekte, findet die Zwischenpräsentation der Projekte an der Hochschule statt. Zu dieser Zwischenpräsentation sind alle Teams, aber explizit nicht die Auftraggeber eingeladen. Die Teams berichten den Dozent_innen und den anderen Teams über den Projektauftrag, die Zusammenarbeit mit dem Auftraggeber, die interne Zusammenarbeit und insbesondere über den aktuellen Projektstand und die geplanten Arbeiten bis zum Projektende. Die Teams sind angehalten, ihre Präsentationen so aufzubauen, als ginge es um eine Anschlussfinanzierung ihres Projektes. Entsprechend kritisch fallen die Nachfragen der Dozent_innen aus, die dabei als potenzielle „business angel“ fungieren.

Die Endpräsentation findet Mitte Februar statt. Zur Endpräsentation werden auch die Auftraggeber, die Studentinnen des ersten Semesters des Studiengangs sowie alle Interessierte der Hochschule eingeladen. Die Endpräsentation besteht aus zwei Teilen. Zunächst präsentieren sich die Teams in einem kurzen Vortrag und erläutern ihr Projektergebnis. Der zweite Teil hat den Charakter einer Messe. An Ständen führen die Projektteams ihre Projektergebnisse vor. Dazu werden Poster, Flyer und Präsentationen erarbeitet. Die Besucher_innen können die entstandenen Programme ausprobieren. Die Endpräsentation, die einen ganzen Tag dauert, wird mit einer gemeinsamen Feier für den erfolgreichen Projektabschluss abgerundet. Nicht nur für die Studentinnen sondern auch für die Auftraggeber stellt die Endpräsentation einen würdigen und gelungenen Abschluss der Projekte dar.

Der Projektabschluss und die Übergabe der Projektergebnisse von den Teams an die Auftraggeber erfolgt meistens nochmal in individuellen Treffen zwischen Teams und Auftraggebern.

Bewertung

Die Bewertung für die Lehrveranstaltung setzt sich aus mehreren Teilbewertungen zusammen. Zu 15% geht die aktive Beteiligung an der Teamarbeit in die Note ein. Diese lässt sich anhand der Arbeitsnachweise aber auch anhand der Dokumentation in dem für die Veranstaltung verwendeten Projektverwaltungstool Redmine belegen. In Redmine sind alle „tasks“ hinterlegt und von wem diese bearbeitet wurden. Die Zwischenpräsentation geht mit 10% und die Endpräsentation mit 20% in die Note ein. Ein gemeinsamer Ergebnisbericht, der nur die erzielten End- und Teilergebnisse auflistet und kurz erläutert sowie die zu erstellende Webseite über das Projekt (eine kurze Seite nach einem vorgegebenen Template) gehen zu 15% in die Bewertung ein. Das eigentliche Projektergebnis wird mit 40% gewichtet. Prinzipiell erhalten die Teammitglieder eines Teams ähnliche Noten. Es werden jedoch Abweichungen in der Mitarbeit und der Effektivität berücksichtigt.

Beispiele aus den letzten Jahren

In den letzten vier Jahren waren folgende Firmen bzw. öffentlichen Einrichtungen (in alphabetischer Reihenfolge) an Projekten beteiligt: adesso AG, Bund der Szenografen e.V., Capgemini Deutschland GmbH, Community Impulse Initiative e.V., Computacenter, CSC Deutschland Solutions GmbH, EAW Relaistechnik GmbH, Exozet Berlin GmbH, Frauen Computer Zentrum Berlin e.V., Fraunhofer IPK, HTW Berlin, InMediasP GmbH, Kulturkosmos Müritz e.V.,

Kunsthochschule Weißensee, Lansinoh Laboratories Inc., mikado AG, Optimal Systems GmbH, PSI, SAP, SHe Kommunikation!, Senatsverwaltung für Stadtentwicklung Berlin, SER, Stephanus IT GmbH, transentis management consulting GmbH & Co. KG, TripsByTips GmbH, QMethods - Business & IT Consulting. Einige der Firmen haben inzwischen mehrere Projekte begleitet.

Projektthemen betrafen u.a. die Erstellung von Web2.0-Seiten (Blogs, Portallösungen, Webshops) unter Verwendung von Content Management-/ Redaktionssystemen, das Ausprobieren und Anwenden neuer Technologien (z.B. HTML5 für die Offlinespeicherung von Webinhalten, für die Erstellung von eBooks und zur Verwaltung von Ressourcen) sowie die Konzepterstellung nach Anforderungserhebung und –analyse inklusive Prototyping. Die Projektergebnisse wurden häufig unter Anwendung agiler Methoden erstellt. Eine Auflistung der Auftraggeber und ihrer Themen befindet sich unter http://fiw.htw-berlin.de/studieren/projekte/.

Motivation und Erwartungshaltung der Beteiligten

Aus unterschiedlichen Gründen sind alle Beteiligten bei der Lehrveranstaltung „Projekte in der Wirtschaft“ hoch motiviert. Auch die Erwartungshaltungen der Einzelnen ist groß. Die Unterschiede und Gründe für die jeweilige Motivation und die wesentlichen Erwartungen der an der Veranstaltung Beteiligten werden im Folgenden diskutiert.

Auftraggeber

Von den Auftragebern wird erwartet, dass sie eine Projektidee entwickeln, diese in einem vorgegebenen Steckbrief beschreiben und zu Beginn des Semester vor Ort präsentieren. Für die Laufzeit des Projekts muss eine Kontaktperson gestellt werden. Diese schlüpft gegenüber den Studentinnen in die Rolle des Auftraggebers/Kunden. Sie steht für die Anforderungserhebung, laufende Abstimmungen während und die Abnahme am Ende des Projekts zur Verfügung. Der Aufwand für die Projektbegleitung ist abhängig von der Aufgabenstellung, mindestens sollten jedoch 3-4 Termine (Kickoff, Abstimmung der Anforderungen, ..., Abnahme) eingeplant werden. Die meisten Auftraggeber treffen sich mit ihrem Team jedoch meistens 14-tägig, häufig sogar wöchentlich. Darüber hinaus und erneut abhängig von der Projektaufgabe sind die Firmen aufgefordert, benötigte Arbeitsmittel (Software, Hardware, Zugänge, Dokumentationen) bereitzustellen und auch für technische Fragestellungen zur Verfügung zu stehen.

Das Interesse der Firmen bzw. öffentlichen Einrichtungen oder Vereine zur Beteiligung an einer solchen Veranstaltung ist groß. Die Motivation dafür ist verschieden. Bei den Firmen spielt sicher in erster Linie die Suche nach geeignetem Personal eine Rolle. Die Projekte bieten ihnen die Möglichkeit, 5-7 Studentinnen über einen längeren Zeitraum kennenzulernen. Zudem arbeiten sich die Studentinnen in dieser Zeit schon in ein Themenfeld des Auftraggebers ein.

Selbst wenn kein fertiges Produkt erstellt wird, bietet die Veranstaltung viel Potential, im Anschluss weiter zusammenzuarbeiten. Die beteiligten Studentinnen kommen aus dem dritten oder dem fünften Fachsemester. Im vierten Semester müssen die Studentinnen ein 17-wöchiges Praktikum ableisten. Im sechsten Semester schreiben sie ihre Bachelorarbeit. Beides gibt den Firmen die Möglichkeit, ausgewählte Studentinnen weiter zu binden und an die im Projekt erarbeiteten Ergebnisse anzuknüpfen.

Eine weitere Motivation besteht darin, neue Technologien auszuprobieren oder innovative Ideen umzusetzen. Diese Ideen bestehen oft schon länger in den Köpfen der Mitarbeiter, wurden aber auf Grund des drängenden Tagesgeschäfts immer wieder verschoben. Die Projekte bieten die Möglichkeit, mit relativ geringem Personaleinsatz solche Ideen aus den Schubladen zu befreien, deren Machbarkeit zu prüfen und (prototypisch) umzusetzen. Aus einzelnen dieser Ideen sind in den vergangenen Jahren Produkte entstanden, die mittlerweile zum Portfolio des Unternehmens gehören.

Bei den öffentlichen Einrichtungen und Vereinen steht das Ergebnis und dessen Einsatz im Vordergrund. Hier sind es oft fehlende Mittel und mangelnde IT-Kenntnisse, die durch den Einsatz von studentischen Teams kompensiert werden sollen.

Obwohl wir bei der Anbahnung der Projekte nicht versprechen können, dass die Ergebnisse in jedem Falle verwendbar sind, zeigt die Vergangenheit, dass viele Projekte erfolgreich und die erarbeiteten Ergebnisse direkt einsetzbar sind. Wenn kein fertiges Produkt entsteht, so ist zumindest das Verständnis für die gewünschte Lösung gewachsen und die Kosten für die finale Umsetzung sind abschätzbar.

Studierende

Durch die Betreuung der Veranstaltung merken wir Professor_innen, dass anfangs (besonders bei den Drittsemestlerinnen) der Respekt vor dem Sprung ins teilweise noch sehr kalte Wasser groß ist. Dem gegenüber steht eine sehr hohe Motivation, die durch die echte Aufgabenstellung und die realen Auftragegeber hervorgerufen wird. Sobald die Studentinnen sich hineinknien und merken, dass Sie das Problem verstehen und Lösungen erarbeiten können ist der Bann gebrochen. Die Identifikation mit dem Projekt und die Bereitschaft gemeinsam eine Lösung zu erarbeiten ist dann sehr groß. Am Ende steht oft eine brauchbare Lösung, vor allem aber sehr viel Stolz und ein gewachsenes Selbstverständnis für die zukünftige berufliche Rolle.

Die Motivation, sich gegenüber Vertretern der Wirtschaft (und somit potenziellen zukünftigen Arbeitge-

bern) erfolgreich zu präsentieren, hilft den Studentinnen den sehr hohen Arbeitsaufwand zu absolvieren, der für die erfolgreiche Durchführung der Projekte geleistet werden muss. Ein wöchentliches Arbeitspensum von 20 Zeitstunden nur für die Projekte sind keine Seltenheit bei den Studentinnen. Gerade in Anbetracht dessen, dass sowohl im dritten als auch im fünften Semester noch jeweils vier weitere Veranstaltungen absolviert werden müssen, ist dieser Aufwand nur mit einer hohen Motivation leistbar. Im dritten Semester kommt für die Studentinnen häufig die Erwartung hinzu, sich mit einer guten Leistung innerhalb des Projektes für ein Praktikum (welches im vierten Semester durchgeführt wird) empfehlen zu können. Die Studentinnen aus dem fünften Semester hingegen setzen nicht selten das Thema des Projektes in ihrer Bachelorarbeit im fünften Semester fort.

Lehrende

Wir Professor_innen erfahren eine Steigerung der eigenen Reputation innerhalb der Hochschule aufgrund der vielen Unternehmenskontakte und -kooperationen. Darüber hinaus profitieren wir und unsere anderen Lehrveranstaltungen von den intensiven Einblicken in die unternehmerische Praxis und in die derzeitig behandelten Themen innerhalb der IT-Branche.

Weitere Motivationen ziehen wir aus den positiven Rückmeldungen der Studierenden, die sich häufig begeistert darüber zeigen, bereits so früh und so intensiv in Kontakt mit der Wirtschaft treten zu können. Im Rahmen der Projekte findet für uns ein interessanter Rollenwechsel weg vom Dozierenden hin zum Lerncoach statt. In Kombination mit der starken Einbindung der Auftraggeber wird der zu leistende Aufwand unter Anbetracht des großen Kompetenzgewinns für die Studentinnen vertretbar.

Es muss betont werden, dass der Aufwand nur deshalb geleistet werden kann, weil alle Professor_innen des Studiengangs intensiv an der Organisation und Durchführung der Projekte beteiligt sind. Dies beginnt bei der Akquise der Auftraggeber. Eine Aufteilung der Projekte auf die drei Professor_innen des Studiengangs führt zu einer Betreuung von 3-4 Projekten pro Professor_in. Gerade in Anbetracht der Vielfältigkeit der Projektthemen und einer dazu notwendigen Einarbeitung der Professor_innen in diese stellt dies auch die Grenze des sinnvoll Machbaren für eine umfassende organisatorische und fachliche Betreuung dar.

Evaluation und Adaption

Die generelle Rückmeldung der Studentinnen ist überwiegend positiv. Die Studentinnen schätzen es, dass zuvor vermitteltes Wissen sehr schnell angewendet wird. Auch wenn vielen Studentinnen am Anfang des Projektes die Kenntnis über die zu verwendenden Technologien fehlen, so erfahren sie doch während der Projektlaufzeit, dass sie das während des Studiums angeeignete Methodenwissen anwenden können, um sich in neue Technologien einzuarbeiten. Die Projekte stellen ein hervorragendes Umfeld dar, Wissen miteinander zu verknüpfen.

Neben den fachlichen Kompetenzen werden Fähigkeiten in der Kommunikation und im Management geschult. Beides wird von den Studentinnen sehr geschätzt. Die Aufgabe, unter Druck in einem Team zusammenzuarbeiten, um eine gemeinsames Ergebnis in einer vorgegebenen Zeit zu erreichen, stellt hohe Anforderungen an die Teamfähigkeit der Studentinnen. Um die Studentinnen noch besser bei der Bewältigung von Teamkonflikten zu unterstützen, wurde bei der Umgestaltung des Curriculums die Lehrveranstaltung Konfliktmanagement parallel zu den Projekten geplant. Wir hoffen dass beide Veranstaltungen in Zukunft davon profitieren: die eine durch reale und aktuelle Beispiele, die andere durch Lösungen für die Zusammenarbeit.

Die Studentinnen empfinden das zielorientierte Arbeiten als sehr effektiv. Dies ist allerdings auch mit einem hohen Leistungsdruck verbunden. Der zu leistende Aufwand wird zumindest während der Projektlaufzeit als zu hoch empfunden. Die Identifikation mit dem Projekt und die Bereitschaft, gemeinsam eine Lösung zu erarbeiten ist jedoch sehr groß. Ist das Projekt erfolgreich geschafft, ist bei den Studentinnen viel Stolz und ein gewachsenes Selbstbewusstsein spürbar.

Erfahrungen aus den letzten Jahren

Die Lehrveranstaltungen wird jedes Jahr anonym evaluiert. Aus den Ergebnissen lassen sich sowohl Erfolg als auch Verbesserungsbedarf der Veranstaltungen ableiten.

Auf die Frage “Was gefällt Ihnen besonders gut bei diesem Lehrangebot?“ äußerten die Studentinnen u.a.: „Praktischer Bezug des Stoffes des Studiums wird vorgestellt. Dies ist sehr motivierend. Auch regt es dazu an Dinge zu lernen, die nicht Teil des Studiums sind.“; „Man lernt wozu man studiert und welchen praktischen Nutzen das Erlernte hat“; „Teamwork unter den Studentinnen“ ; „per Selbststudium viel gelernt“ ; „Teamfähigkeit gestärkt“ ; ...

Um das Verbesserungspotential zu erkennen werden die Studentinnen gefragt: „Was kann besser werden?“ Darauf wurde immer wieder genannt: „bessere Wahl der Auftraggeber“, „mehr Zeit für die Projektarbeit während des Semesters (d.h. weniger weitere Lehrveranstaltungen)“ und „bessere Information der Auftraggeber über den Kenntnisstand der Studentinnen (3. Semester!)“.

Die Kritik, dass solche Veranstaltungen erst später im Studium kommen sollten, haben wir bisher nicht gehört. Auch die Wiederholung im 5. Semester wird geschätzt. Mögliche Probleme werden eher erkannt und der Umgang mit ihnen geschieht viel offensiver,

da man Fehler aus dem vorherigen Durchlauf nicht wiederholen möchte.

Aus den Erfahrungen der letzten Jahre und dem Feedback der Studentinnen wurden bisher folgende Änderungen vorgenommen.

Wahl der Auftraggeber/Projekte: Die Auftraggeber erhalten zu Beginn einen Handzettel, der die Erwartungen an die Auftraggeber beschreibt, Vorteile für die Beteiligten aufzeigt und über den Kenntnisstand der Studentinnen sowie typische mögliche Aufgabenstellungen informiert. Darüber hinaus besteht die Möglichkeit, Einsicht in bestehende Steckbriefe zu erhalten.

In einem Durchlauf wurde ein Projekt mehr als nötig akquiriert. Die Idee war, dadurch etwas Flexibilität zu gewinnen, falls ein Projekt von niemanden gewollt wird. Davon sind wir mittlerweile wieder abgerückt, da das Resultat nicht das Gewünschte war. Ein unserer Meinung nach fachlich und technologisch sehr passendes Entwicklungsprojekt wurde zugunsten eines weniger gut vorbereiteten Projekts abgewählt, nur weil bei Letzterem der Auftraggeber den prominenteren Namen hatte.

Zeit für die Projekte: Der Zeitfrage wurde bei der Neugestaltung/Reakkreditierung des Curriculums Rechnung getragen. So wurden die Credits für die Projektveranstaltungen von 5 Credits auf 10 Credits erhöht.

Teamzusammenstellung: Insbesondere bei der Teamzusammensetzung und bei der Zuordnung der Projekte haben wir in den letzten Jahren unterschiedliche Varianten ausprobiert. So wurde in einem Jahr erlaubt, dass sich die Teams selbständig bilden. Dies erwies sich aus mehreren Gründen als Nachteil. Einerseits fanden sich oft Studentinnen mit einem ähnlichen Leistungsniveau zusammen. Dies hatte zur Folge, dass es sehr leistungsstarke und leistungsschwache Teams gab. Außerdem wurden verstärkt Teams mit einem gemeinsamen Migrationshintergrund gebildet, d.h. es gab „türkische“ und „russische“ Teams. Trotz der eigenen Wahl der Teamzusammenstellung zeigte sich am Ende jedoch kein geringeres Konfliktpotenzial als bei einem reinen Losen der Teams. Vielmahr hat sich gezeigt, dass durch Losen entstandene Teams deutlich professioneller und formaler miteinander arbeiten. Es hat sich für den Projekterfolg als ein großer Vorteil erwiesen, dass Teams nun hinsichtlich des Leistungsvermögens und hinsichtlich des kulturellen Hintergrundes besser gemischt sind.

Ein außerordentlicher Gewinn war die Entscheidung, die Teams auch über die Semestergrenzen hinaus zu mischen. Früher gab es Projekte nur für die Drittsemester und nur für die Fünftsemester. Das hatte auch zur Folge, dass die Auftraggeber lieber Projekte mit den Studentinnen aus dem fünften Semester durchführen wollten und die Projekte für die Drittsemester nicht so anspruchsvoll waren. Die jetzige Zusammensetzung der Teams aus jeweils gleich vielen Dritt- und Fünftsemestern sorgt für einen besseren Austausch der Erfahrungen unter den Studentinnen und bietet eine unternehmensgleiche Projektstruktur von Senior- und Junior-Projektmitarbeiterinnen. Insbesondere aus den Erfahrungen der Fünftsemester über die Projektorganisation und über die Kommunikation mit dem „Kunden“ profitieren die Studentinnen des dritten Semesters.

Zuordnung der Projekte: Hier wurde bisher am meisten experimentiert. In den ersten Jahren durfte sich jede Studentin einzeln für ein Projekt entscheiden. Dafür wurden alle Projektnamen nebeneinander an die Tafel geschrieben. Auf ein Signal hin wurden die Studentinnen gebeten nach vorne zu kommen und ihren Namen unter das Projekt ihrer Wahl zu schreiben. Dabei durften einem Projekt maximal sechs Namen zugeordnet werden.

An und für sich hat diese Art der Zuordnung gut funktioniert: die Studentinnen, die genau wissen, in welches Projekt sie wollen, kommen sehr zielstrebig nach vorn, um ihren Namen einem Projekt zuzuordnen; die etwas Unentschlossenen (und meist auch Langsameren) orientieren sich dann daran, mit wem sie gerne zusammenarbeiten wollen. Aufgefüllt werden die Projekte dann durch diejenigen, die keine vorgefertigten Prioritäten haben. Ging die Verteilung trotz des gewählten Verfahrens nicht auf, wurde zuerst versucht, die Einigung den Studentinnen zu überlassen. Erst wenn dies nicht gelang, haben die Lehrenden steuernd eingegriffen. Bewährt hatte sich hier auch eine vorher gezogene Losnummer. Diese konnte verglichen werden, um den Anspruch einer Gruppe auf ein Projekt zu prüfen. Nachteil dieses Verfahrens war die schon genannte Homogenität der entstandenen Gruppen.

Im letzten Jahr wurden die Team erstmals über die Semestergrenzen hinweg gelost. Die Zuordnung der Teams zu den Projekten wurde dann den Teams überlassen, wobei allen Teams per Los ein „Defaultprojekt“ zugeordnet wurde, welches anschließend zurückgegeben oder getauscht werden konnte. In diesem Jahr wurde dann erstmals die Methode mit den drei Wahlgängen erprobt, die (zumindest dieses Jahr) für eine allseits akzeptierte Verteilung der Projekte gesorgt hat.

Zusammenfassung

Der Mehrwert der „echten“ Projekte für die Studentinnen ist offensichtlich. Sie gewinnen einen Einblick in reale Aufgabenstellungen, lernen nebenbei echte Firmen und deren Ansprechpartner kennen. Da sie für die Bearbeitung der Projekte oft vor Ort sind, bekommen Sie auch einen Einblick in die Arbeitsumgebungen, typische Prozesse und die Firmenkultur. Der eigentliche Gewinn liegt aber in der umgreifenden Stärkung ihrer Kompetenzen. Durch die lebendige Lernform arbeiten sie sich wie selbstverständlich in völlig neue Themen und Technologien ein und bauen bzw. verstärken so ihre Fach- und Sachkompetenzen. Durch die selbstorganisierte Teamarbeit und die echte Kunden-Auftragegber Situation werden ausserdem Sozial und Methodenkompetenz trainiert. Die gewonnenen Erfahrungen können sie im Anschluss sowohl beim Studium als auch bei der Bewerbung und im ersten Job einsetzen. Unser Fazit: Eine ähnliche Motivation wie in den Projekten ist durch herkömmliche Lehrveranstaltungen mit oft künstlich arrangierten Aufgaben oder Rollenspielen nicht zu erreichen.

Literatur

[Belbin 2012] BELBIN, Meredith: *Team Roles at Work*. Taylor & Francis, 2012. – ISBN 9781136434822

[Kleuker u. Thiesing 2011] KLEUKER, Stephan ; THIESING, Frank M.: Vier Jahre Software-Engineering-Projekte im Bachelor - ein Statusbericht. In: *Software Engineering im Unterricht der Hochschulen (SEUH)*, 2011, S. 40–44

[Lewerentz u. Rust 2001] LEWERENTZ, Claus ; RUST, Heinrich: Software-Ingenieure als kompetente Teamworker. In: *Softwaretechnik-Trends* 21 (2001), Nr. 1

[Liebehenschel 2013] LIEBEHENSCHEL, Jens: Software-Engineering Projekte in der Ausbildung an Hochschulen - Konzept, Erfahrungen und Ideen. In: *Software Engineering im Unterricht der Hochschulen (SEUH)*, 2013, S. 27–34

[Rolf u. a. 2013] ROLF, A. ; MÖLLER, A. ; FUNK, B. ; NIEMEYER, P.: Freie Pizzawahl für Informatiker und Wirtschaftsinformatiker - Didaktische Herausforderungen für Informatik und Wirtschaftsinformatik angesichts der digitalen Gesellschaft. In: *Informatik Spektrum* 36 (2013), Nr. 1, S. 90–98

[SIBB e.V. 2012] SIBB E.V. - ICT & DIGITAL BUSINESS ASSOCIATION BERLIN-BRANDENBURG (Hrsg.): *Human Capital & Growth-Index (HCG-Index)*. Berlin: SIBB e.V. - ICT & Digital Business Association Berlin-Brandenburg, 2012

[Siegeris u. Krefting 2014] SIEGERIS, Juliane ; KREFTING, Dagmar: Lehrmethodenvielfalt im Curriculum des Studiengangs Informatik und Wirtschaft Ein Erfahrungsbericht. In: LEICHTSCHOLTEN, Carmen (Hrsg.) ; SCHROEDER, Ulrik (Hrsg.): *Informatikkultur neu denken Konzepte für Studium und Lehre*. Wiesbaden : Springer Fachmedien, 2014, S. 127–139. – ISBN 78-3-658-06021-3

[Vigenschow u. a. 2011] VIGENSCHOW, U. ; SCHNEIDER, B. ; MEYROSE, I.: *Soft Skills für Softwareentwickler - Fragetechniken, Konfliktmanagement, Kommunikationstypen und -modelle*. dpunkt.verlag, 2011

Clean Code – ein neues Ziel im Software-Praktikum

Doris Schmedding, Anna Vasileva, Julian Remmers, Technische Universität Dortmund

doris.schmedding | anna.vasileva | julian.remmers@tu-dortmund.de

Zusammenfassung

In diesem Beitrag wird ein Konzept vorgestellt, wie in der Informatik-Ausbildung die Sensibilität für guten Code erhöht werden soll. Das Software-Praktikum bietet einen geeigneten Rahmen, dieses Ausbildungsziel umzusetzen. Wir zeigen Code-Defizite, die sich in studentischen Projekten häufig finden. Es wird ein Tool vorgestellt, das bei der Entdeckung unterstützt, und Refactorings präsentiert, mit denen die Studierenden die selbst gefundenen Defizite leicht beheben können.

Motivation

In den ersten Semestern der universitären Ausbildung liegt der Schwerpunkt darauf, die Konzepte der verwendeten Programmiersprachen kennen zu lernen und funktional korrekte Programme zu schreiben. Dies wird zumindest an der Fakultät für Informatik an der TU Dortmund aufgrund der großen Anzahl der Studienanfänger weitgehend durch automatisierte Tests überprüft. In der Software-Technik-Vorlesung liegt der Schwerpunkt auf der Modellierung, Mustern [Gamma 2004] und dem Vorgehen in Software-Entwicklungsprojekten, weniger auf der Programmierung.

Erst in einem studentischen Software-Entwicklungsprojekt kommen die verschiedenen Aspekte der Software-Entwicklung wie Prozessmodell, Modellierung und Programmierung gemeinsam zum Einsatz und fügen sich zu einem Ganzen. Die Bedeutung von Lesbarkeit und Verständlichkeit des Codes kann sich den Studierenden aber erst wirklich erschließen, wenn mit mehreren Studierenden gleichzeitig in einem komplexen Projekten gearbeitet wird und die Studierenden anhand von selbst produzierten Beispielen eigene evtl. auch leidvolle Erfahrungen beim Lesen fremden Codes sammeln. Auch wenn die Wartbarkeit von Programmen ein wichtiges Ziel in der Software-Entwicklung darstellt, lässt sie sich im universitären Kontext, in dem studentische Projekte nur eine kurze Lebensdauer haben, nur schwer vermitteln.

Im Software-Praktikum (SoPra) in unserer Fakultät führen acht BA-Studierende gemeinsam Projekte durch. Etwa sechs bis 10 Gruppen bearbeiten gleichzeitig die gleichen Aufgaben. In einem Blockpraktikum in der vorlesungsfreien Zeit dauert eine Projekt drei Wochen. Es werden nacheinander in 6 Wochen zwei Projekte durchgeführt.

Wir verwenden die Programmiersprache Java. Zur Modellierung setzen wir UML mit dem Tool Astah ein. Aus dem Modell werden Java-Code-Rahmen generiert. Als Entwicklungsumgebung benutzen wir Eclipse mit dem SVN-Plugin Subclipse. Die Dokumentation mit JavaDoc und JUnit-Tests sind wichtige Bestandteile unseres Entwicklungsprozesses und inzwischen ebenso wie alle oben genannten Tools und Methoden allgemein akzeptiert. Am Ende eines Projekts erfolgt die Abnahme und Bewertung der Projekte gegenseitig durch die Gruppen. Die Ergebnisse der Bewertung werden im Plenum diskutiert.

Auch wenn man nach drei Wochen nicht erwarten kann, ein perfekt funktionierendes Programm zu erhalten, wird die funktionale Korrektheit als Ziel in der Software-Entwicklung von keinem der Beteiligten in Frage gestellt. Für die Studierenden ist in ihrem eigenen Projekt und bei der Bewertung der Ergebnisse der anderen Gruppen auch das Aussehen des Programms sehr wichtig, insbesondere bei Spielen. Bisher überhaupt nicht im Fokus stand die innere Qualität der Programme. Das wollen wir ändern. Unsere Vision ist, dass der Qualitäts-Check nach einer Änderung genauso selbstverständlich wie der JUnit-Test wird.

Vorgehensweise

Dazu identifizieren wir zunächst einmal typische Defizite in studentischen Projekten [Remmers 2014]. Aus der Masse von Qualitätsmängeln, die Martin [Martin 2008] beschreibt, wählen wir diejenigen aus, die

- für Studierende leicht verständlich sind,

- häufig in Studenten-Projekten vorkommen,
- leicht und möglichst mit Tool-Unterstützung erkennbar sind und
- mit Tool-Unterstützung einfach zu beheben sind.

Ab wann ein Qualitätsmangel vorliegt, ist eine Frage des persönlichen Anspruchs an die Qualität. Wenn man mit einem Tool Qualitätsmessungen durchführt, lassen sich Grenzwerte angeben, was noch toleriert wird und ab wann ein Defekt vorliegen soll. Wir geben jeweils an, für welche Grenzen wir uns entschieden haben und diskutieren ihre Sinnhaftigkeit.

Ein erster Einsatz unserer Messinstrumente und Grenzwerte im Software-Praktikum in der vorlesungsfreien Zeit nach dem Sommersemester 2014 diente zur Evaluation, ob die angenommenen Mängel tatsächlich mit Hilfe des von uns ausgewählten Tools festzustellen sind und in welchem Umfang (siehe "Ergebnisse unserer Messungen") sie auftreten. Am Ende des SoPras erhielten die Gruppen jeweils einen Bericht über die festgestellten Mängel.

In einem zweiten Einsatz im SoPra im WS 14/15 ist geplant, bereits in der Einführungsveranstaltung das Thema Code-Qualität anhand einiger Folien einzuführen und die Qualitätsmessung anzukündigen. Den Studierenden stehen im Wiki des Software-Praktikums Tutorials zum Thema Clean Code [Martin 2008], statische Code-Analyse und Refactoring [Fowler1994] für das Selbststudium zur Verfügung. Außerdem wird bei der Eclipse-Einführung gezeigt, wie einfache Refactorings durchgeführt werden. Im Rahmen der Reflexion der Erfahrungen im ersten Projekt [Schmedding 2011] soll jede Gruppe nach dem ersten Projekt einen Bericht über die festgestellten Mängel erhalten und eine Diskussion über die möglichen Ursachen und Folgen der Mängel geführt werden. Wir erwarten im zweiten Projekt eine Verbesserung der Code-Qualität.

Durch die Beschränkung auf zunächst einmal wenige Mängel und den Einsatz eines Tools zur Qualitätskontrolle, jeweils gekoppelt mit Hinweisen, wie man den Mangel mit Tool-unterstützten Refactorings einfach beheben kann, erhoffen wir uns eine große Akzeptanz unserer Qualitätsoffensive.

Clean Code

In [Matin 2008] werden sehr viele Qualitätsmängel beschrieben, die sehr ambitioniert sind und sich den Studierenden zum Teil nur schwer vermitteln lassen, z. B. dass eine Methode maximal vier Zeilen lang sein soll. Jedenfalls erschienen uns Martins Qualitätsanforderungen für unsere Studierenden völlig unerreichbar, so dass wir einige erreichbare Regeln [Remmers 2014] definiert haben, die von den SoPra-TeilnehmerInnen nicht verletzt werden sollten.

Die Regeln wurden in die folgenden drei Gruppen eingeteilt:

- **Bezeichner** sollen verständlich sein und die Java-Konventionen einhalten. Humor ist bei der Wahl der Bezeichner zu vermeiden!
- **Methoden** sollen nicht zu komplex sein, eine bestimmte Maximallänge nicht überschreiten und wegen der Vertauschungsgefahr nicht zu viele Parameter haben.
- **Klassen** sollen nicht zu lang sein, keinen toten Code enthalten und keine Gott-Klassen sein.

Gott-Klassen oder Gott-Objekte [Riel 1996] gehören zu den sogenannten Anti-Patterns und widersprechen dem „Single-Responsibility"-Prinzip [Martin 2008]**.** Als Gott-Klassen werden Klassen bezeichnet, deren Methoden zu komplex sind, die auf zu viele Attribute anderer Klassen zugreift und deren Methoden nur eine geringe oder gar keine Kohäsion besitzen.

Außerdem haben wir nach dem Anti-Pattern „Magische Werte" („Magic Values") und nach Literalen anstelle von Konstanten in Bedingungen suchen lassen, da sie die Lesbarkeit und Wartbarkeit der Bedingungen verletzen.

Statische Programmanalyse

Wir setzen die statische Programmanalyse hier im Software-Praktikum zur Aufdeckung von Qualitätsmängeln ein. Dazu verwenden wir das Eclipse-Plugin PMD[1]. PMD prüft die Einhaltung einer sehr umfangreichen Menge von vordefinierten Regeln, die jeweils unter bestimmten Oberbegriffen zusammengefasst sind. Manche der von PMD festgelegten Qualitätsregeln empfinden wir als zu streng, z. B. die Forderung, dass eine Methode nur ein „return statement" enthalten soll. Dagegen sollten unserer Meinung nach manche Grenzwerte wegen des relativ kleinen Projektumfangs und der begrenzten Projektlaufzeit niedriger liegen.

[1] http://sourceforge.net/projects/pmd/files/pmd-eclipse/

Der Benutzer kann unter `Preferences->PMD` einzelne Regeln oder Regelgruppen an- oder abwählen. Außerdem besteht die Möglichkeit, eine spezielle Regelmenge in Form einer XML-Datei (siehe Anhang B) zu laden. Für diesen Weg haben wir uns entschieden, um alle Studierenden mit den gleichen auf die Lehrveranstaltung zugeschnittenen Qualitätsmaßstäben zu versorgen. Wie bereits erläutert, haben wir für die Ausbildung im Software-Praktikum eigene Grenzwerte definiert. Auch Spillner und Linz empfehlen in [Spillner 2005], beim erstmaligen Einsatz eines Analysetools die Grenzwerte so zu wählen, dass die Akzeptanz eines derartigen Tools nicht gefährdet wird.

Wir haben PMD als Analysetool ausgewählt, weil es flexibel ist und gut in unsere Arbeitsumgebung passt. Außer PMD können die Studierenden auch das Eclipse-Plugin FindBugs[2] zur statischen Code-Analyse benutzen, das ebenfalls Bestandteil der vorbereiteten Entwicklungsumgebung ist.

Im SoPra-Wiki wird für das Selbststudium gezeigt, wie man PMD (und FindBugs) verwendet und eigene Regeln einbindet. Die im SoPra eingesetzten PMD-Regeln sind in Anhang B im XML-Format aufgelistet. Die gewählten konkreten Grenzwerte für die eingesetzten Zählmetriken kann man dem Anhang A entnehmen. Die Wahl der Grenzwerte wird jeweils im Zusammenhang mit den Ergebnissen der Messungen erläutert. Einige Grenzwerte haben wir von PMD übernommen, für andere haben wir unserem Ausbildungsumfeld angepasste Werte gewählt. Das Kriterium für die Erkennung einer möglichen Gott-Klasse wird im Zusammenhang mit den Messergebnissen (siehe Klassen) vorgestellt und diskutiert.

Nicht alle für das Software-Praktikum als wichtig ausgewählten Regeln des Clean Codes lassen sich automatisch überprüfen. Kein Tool kann feststellen, ob ein Bezeichner sinnvoll gewählt wurde. Als Indiz für eine sorgfältige Wahl dient uns die Länge eines Bezeichners, die wir messen können.

Aber wie Spillner und Linz [Spillner 2005] sind wir der Ansicht, dass sich Qualitätsstandards und Konventionen nur dann einführen und beibehalten lassen, wenn geeignete Prüfwerkzeuge vorhanden und leicht zu bedienen sind, da ein nicht automatisch überprüfbares Regelwerk nur als bürokratischer Ballast empfunden wird.

Refactoring

Sogenannte „Refactorings“ [Fowler 1999], Verbesserungen bestehenden Codes, werden eingesetzt, um die mit PMD gefundenen Mängel zu beseitigen. Wichtig ist, dass die Funktionalität erhalten bleiben muss, was durch ständige Tests überprüft wird. Viele Refactorings sind in unserer Entwicklungsumgebung Eclipse bereits standartmäßig integriert, so dass sie über das Kontextmenü leicht zu erreichen und anzuwenden sind.

Mit Hilfe von Tutorials im SoPra-Wiki für das Selbststudium wird gezeigt, welche Refactoring-Techniken die Studierenden einsetzen können, um die mit PMD (und FindBugs) gefundenen Mängel zu beseitigen. Ähnlich wie Martin beschreibt Fowler in [Fowler 1999] eine Fülle von Verbesserungen (~70), aus denen wir diejenigen ausgewählt haben, die wir für unsere Studierenden für praktikabel und gut nachvollziehbar halten. Diejenigen, die sich mit Tool-Unterstützung umsetzen lassen, sind die folgenden:

- **Rename** wird zur Verbesserung der Namensgebung eingesetzt und kann auf alle Bezeichner angewendet werden. Da alle Aufrufe mit geändert werden, können die Studierenden gut die Mächtigkeit des Werkzeugs erkennen.
- **Extract Method** kann eingesetzt werden, wenn eine Methode zu lang oder zu komplex geworden ist. Das Werkzeug legt für den markierten Bereich eine neue Methode an und ersetzt ihn durch den Aufruf der Methode.
- **Extract Local Variable** kann verwendet werden, wenn der Code aufgrund einer langen Aufrufkette nur noch schwer lesbar ist. Dann wird für die markierte Aufrufkette automatisch eine neue lokale Variable angelegt.
- Mit **Introduce Parameter Object** wird eine überlange Parameterliste durch ein Parameterobjekt ersetzt. Zusätzlich wird eine neue Klasse mit den früheren Parametern als Attributen angelegt, auf Wunsch auch mit entsprechenden Zugriffsmethoden.
- Mit **Extract Constant** kann eine Zahl in einer Bedingung in eine Konstante umgewandelt werden, um die Wartbarkeit und Lesbarkeit des Codes zu erhöhen.
- **Move** verschiebt eine Komponente einer Klasse innerhalb einer Klasse, bei Bedarf auch in eine andere Klasse, sofern die ursprüngliche Klasse eine Referenz auf die neue Klasse besitzt. Dann werden in Falle von Methoden ihre Aufrufe entsprechend geändert.

[2] http://findbugs.sourceforge.net/

Alle anderen Mängel sollten manuell, also ohne Eclipse-Refactoring-Tool, behoben werden. Das ist z. B. die Beseitigung von totem Code kein Problem. Von der Anwendung von **Extract Class** in Eclipse raten wir ab, da dessen Funktionalität nicht der in [Fowler1999] beschriebenen entspricht. **Extract Class** wird eingesetzt, um eine zu komplex oder zu lang geratene Klasse zu teilen. Auch **Extract Method** kann nicht immer voll automatisch vom Tool durchgeführt werden. Wenn in der zu extrahierenden Anweisungsfolge mehrere Werte geändert werden, stößt die Code-Transformation von Eclipse an ihre Grenzen, die man auf jeden Fall berücksichtigen sollte.

Eine wirkliche Verbesserung des Codes erhält man in der Regel erst, wenn man eine Kombination von Refactoring-Techniken anwendet, z. B. erst **Extract Class** und danach **Move**. Selbstverständlich muss nach jeder Veränderung geprüft werden, ob die verlangte Funktionalität noch gegeben ist und ob die Mängel wirklich beseitigt wurden oder neue Mängel entstanden sind.

Ergebnisse unserer Messungen

In diesem Kapitel stellen wir die Ergebnisse unser Messungen vor, die wir durchgeführt haben, ohne dass die Studierenden zuvor informiert waren, dass wir die Codequalität analysieren würden.

Aufgabe in diesem Projekt war die Realisierung eines Computerspiels, das Brettspiel Cartagena[3] in den Varianten Jamaika und Tortuga. Das Spiel sollte mit folgenden Anforderungen realisiert werden:

- Drei verschiedene simulierte Gegner,
- Rücknahme der Züge bis zum Spielbeginn,
- Speichern und Laden angefangener Spiele,
- Highscore-Liste,
- Tipp für unerfahrene SpielerInnen,
- Editor für die freie Gestaltung des Spielfelds,
- Einlesen einer Spielsituation im vorgegebenen Format.

Am Software-Praktikum haben 8 Gruppen mit je 8 Mitgliedern teilgenommen. Alle Gruppen haben funktionstüchtige Spielprogramme produziert, die die oben aufgelisteten Anforderungen erfüllt haben. Da eine Gruppe (Gruppe 6) nicht unseren SVN-Server sondern Git[4] zur Versionsverwaltung benutzt hat, konnten wir keine Messung vornehmen. Eine Gruppe (Gruppe 1) ist nicht zustande gekommen. Abb. 1 und Abb. 2 zeigen deshalb den Umfang der Projekte von sieben Gruppen.

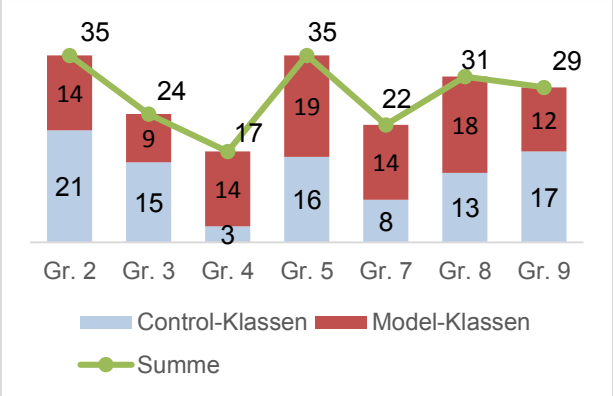


Abb. 1: Anzahl der Klassen

Die GUI-Klassen haben wir bei der Umfangssmessung nicht mit berücksichtigt, da sie zwar sehr umfangreich sind, ihr Programmierstil aber weitgehend von der Game-Engine (Slick2D oder LibGDX) geprägt ist. Wir interessieren uns mehr für die Teile der Programme, welche nach der Modellierung mit UML selbst entwickelt wurden.

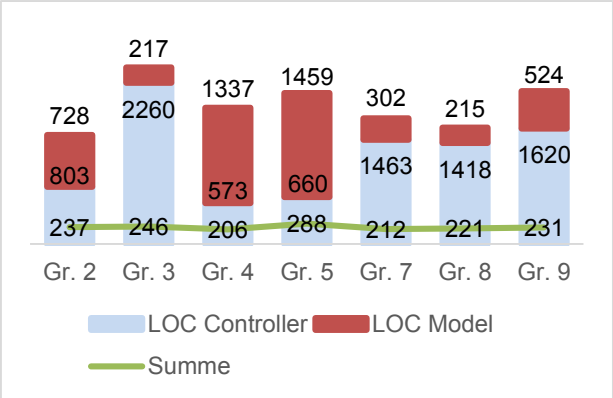


Abb. 2: LOC differenziert nach Model- und Control-Klassen und Anzahl der Methoden insgesamt

Man sieht, dass die Projekte sowohl in der Anzahl der Klassen (Abb. 1), zwischen 17 und 35, als auch im Umfang (Abb.2), zw. 1531 und 2477 LOC (Lines of Code), und Methodenanzahl, zw. 206 und 288, stark schwanken.

Da wir das MVC-Muster [Gamma u.a. 2004] für die Strukturierung der Software vorgeben, wird in Abb. 1 und Abb. 2 zwischen Model- und Control-Klassen unterschieden. Die beiden Gruppen 4 und 5 haben eher dem objektorientierten Paradigma folgend die Programmlogik den Klassen des Problembereichs, den Model-Klassen, zugeordnet, während die Model-Klassen Gruppen 3, 7, 8 und 9 fast nur Datenhaltung betreiben. Das wird in Abb. 2 gut deutlich: Die Model-Klassen von Gruppe 4 und

[3] http://de.wikipedia.org/wiki/Cartagena_(Spiel)

[4] http://git-scm.com/

5 sind deutlich umfangreicher als ihre Control-Klassen und die Model-Klassen der übrigen Gruppen. Wir werden später sehen, dass, wie zu erwarten war, die einzelnen Model-Klassen der anderen Gruppen, die rein der Datenhaltung dienen, nur kurz und wenig komplex sind. Das gilt auch für die Methoden dieser Klassen. Die Gruppe 2 hat während des Projektverlaufs ihre Strategie verändert, von zunächst „ganz dummen" Datenklassen, zu Model-Klassen mit mehr Logik. Daher rührt das relativ ausgeglichene Bild vom Umfang von Model- und Control-Klassen in Abb. 2.

Am Ende der Projektlaufzeit von drei Wochen haben wir ein Code-Review vorgenommen, indem wir die verfügbaren Projekte der Gruppen und die von PMD angezeigten Defekte kritisch analysiert und dokumentiert haben. Dazu mussten wir die Projekte aus dem jeweiligen SVN-Repository herunterladen und die Messungen mit PMD und den SoPra-spezifischen Regeln durchführen. Die Stellen, an denen PMD Defekte gefunden hat, haben wir uns genauer angesehen und selbst beurteilt, ob es sich um einen relevanten Verstoß handelt. Pro Gruppe haben wir zu Zweit etwa eine Stunde benötigt.

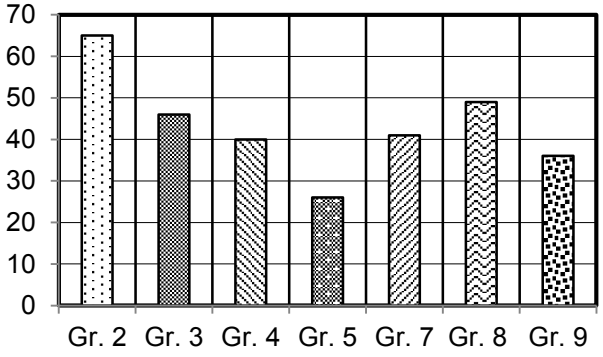


Abb. 3: Gesamtergebnis

Zum Projektabschluss bewerten die Gruppen gegenseitig ihre Projekte anhand eines von uns vorgegebenen Schemas, das in erster Linie die geforderte Funktionalität und die Benutzungs-freundlichkeit berücksichtigt. Analog dazu wurde ein Bewertungsschema (siehe Anhang A) mit Punkten entwickelt, um die Gruppe (mit Schokolade) zu belohnen, die den nach unseren Erkenntnissen besten Code geschrieben hat. Abb. 3 zeigt die Summe der jeweils erreichten Punkte. Gruppe 2, die Siegergruppe, hat 65 von 75 möglichen Punkten erreicht. Zusätzlich bekam jedes Entwicklungsteam einen Bericht, welche Mängel in ihrem Projekt besonders typisch und häufig waren.

Nachfolgend werden die verschiedenen Aspekte diskutiert, die in die Bewertung eingegangen sind. Für jede Kategorie gab es maximal 10, Punkte pro gefundenen Verstoß wurde ein Punkt abgezogen. Da wir davon ausgegangen sind, dass extrem lange Klassen im SoPra selten sind, gab es dafür nur maximal 5 Punkte.

Namensgebung

Bei der Suche nach Defekten in der Namensgebung wurden verschiedene Aspekte unterschieden (siehe Abb. 4):

- Nicht-Einhaltungen der Java-Konventionen bei der Bezeichnerwahl. Das kann von PMD festgestellt werden.
- Sinnvolle oder nicht sinnvolle Bezeichner erkennt PMD natürlich nicht. Wir suchen nach besonders kurzen Bezeichnern (min. 5 Zeichen), schauen uns die Fundstellen an und entscheiden über die Verständlichkeit. Bezeichner „t" oder „x" haben bei uns keine Chance und führen zu Punktabzug. Wir akzeptieren dagegen Bezeichner aus unserem Problembereich wie „Game" oder „Zug".

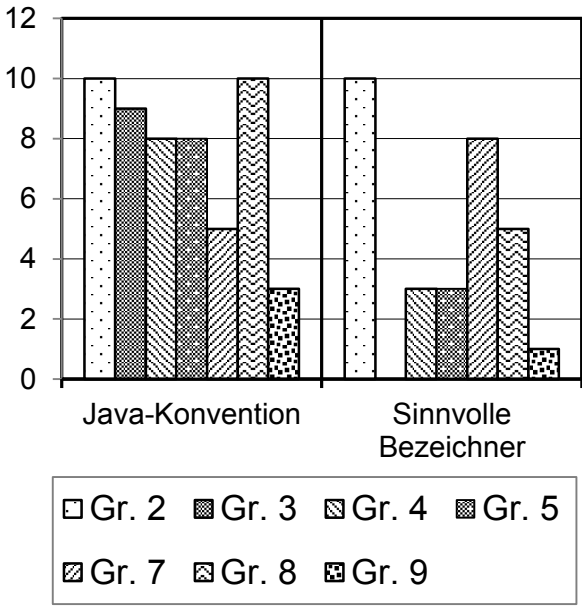


Abb. 4: Namensgebung

Bei Gruppe 2 haben wir keine Defekte in der Namensgebung gefunden, da ein Gruppenmitglied den Code überarbeitet hat. Wir können daraus schließen, dass sich Verstöße bei der Namensgebung leicht mit PMD erkennen und mit den Refactoring-Techniken von Eclipse beseitigen lassen.
Die Java-Konventionen werden von allen Gruppen weitgehend eingehalten. Wir vermuten, dass es daran liegt, dass viele Bezeichner aus dem UML-Modell stammen, das einem mehrfachen Review-Prozess unterzogen wird. Besonders viele einbuchstabige Bezeichner findet man bei den Parametern.

Auffällig waren Unterstriche in Methodennamen in Gruppe 9, die nicht den Java-Konventionen entsprechen (Abb. 4). Sie haben wohl ihre Ursache in dem zuvor besuchten C-Kurs.

Methoden

Bei Methoden haben wir ihre Länge, die Anzahl der Parameter und ihre zyklomatische Komplexität untersucht.

Eine Methode sollte maximal 40 Zeilen lang sein. Bei PMD ist 100 als maximale Länge von Methoden eingestellt. Martin akzeptiert nur 4 Zeilen. Wir denken aber, dass die Studierenden etwa 40 Zeilen noch überschauen können. Außerdem wollten wir die Messlatte auch nicht zu hoch bzw. zu niedrig setzen.

Wie man sieht (Abb. 5), hat es keine Gruppe geschafft, ohne zu lange Methoden auszukommen. Die längste gefundene Methode mit 186 Zeilen hat die Gruppe 9 produziert. Die Gruppe 8 hat so viele lange Methoden geschrieben, dass in diesem Bereich keine Punkte erhalten geblieben sind.

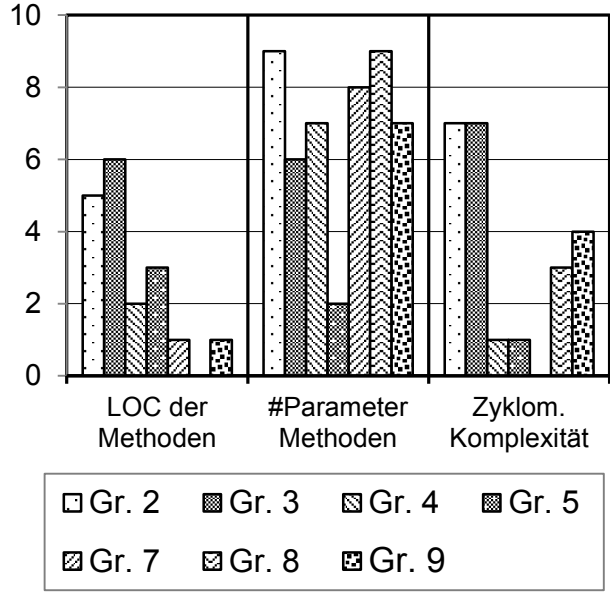


Abb. 5: Methoden

Die extrem langen Methoden besitzen in der Regel auch eine hohe zyklomatische Komplexität. Die längste Methode (von Gruppe 9) hatte eine zyklomatische Komplexität von 49. Als mangelhaft werden Methoden ab einer Komplexität von 10 gewertet. Das entspricht dem von McCabe [McCabe 1976] selbst vorgeschlagenen Wert. Die Methode mit der höchsten zyklomatischen Komplexität von 67 ist „loadGame“ von Gruppe 4.

Auch die von Eclipse generierten equals-Methoden von Klassen sind schwer zu lesen und werden von PMD als zu komplex eingestuft.

Bei der genaueren Betrachtung der besonders komplexen Methoden sieht man deutlich, wie die Entwickler mit der Komplexität „gekämpft“ haben. Man findet Ausgaben auf die Konsole, auskommentierte Programmzeilen, Kommentare zur eigenen Orientierung und Fragen („Was soll ich damit?“). Keine Gruppe hat es geschafft, ohne zu komplexe Methoden auszukommen.

Eine Parameterliste wird als zu lang angesehen, wenn sie fünf oder mehr Parameter beinhaltet. Der Defekt, sehr lange Parameterlisten zu verwenden, ist in den Gruppen unterschiedlich häufig vorgekommen. Im Programm von Gruppe 5 fand sich eine Methode mit 7 Parametern. Das war der höchste gefundene Wert.

Klassen

Wir betrachten Klassen als zu groß, wenn sie mehr als 400 Zeilen lang sind. Voreingestellt bei PMD sind 1000 Zeilen, was bei einem Projektumfang von etwa 2000 LOC im Software-Praktikum viel zu viel ist und in keiner Weise den Ideen von Clean Code entspricht.

Wie bereits erwähnt, haben wir nur 5 Pluspunkte als Guthaben für diese Rubrik verteilt. Alle Gruppen konnten mindestens 2 Punkte behalten (siehe Abb. 6), so dass unsere Vermutung bestätig wurde, dass sehr große Klassen im SoPra selten sind. Gruppe 2, Siegerin der Gesamtwertung und eine der Gruppen mit den meisten Klassen (35), hat überhaupt keine überlange Klasse produziert.

Die zu langen Klassen sind auch die, die von PMD als Gott-Klassen identifiziert werden. Gott-Klassen sind Klassen, die zu viel wissen und sich in zu viele Dinge einmischen.

PMD setzt in der Regel “GodClass“[5] eine Strategie zur Erkennung um, welche in [LMD06] beschrieben wurde. Nach dieser Strategie kann eine Gott-Klasse durch die Berechnung der drei Werte WMC, ATFD und TCC erkannt werden:

- WMC (Weighted Method Count) ist die Summe der zyklomatischen Komplexitäten aller Methoden einer Klasse. Diese darf nach der Regel „GodClass“ den Grenzwert von 47 nicht überschreiten.

[5] Die Regel „GodClass“ wird in folgender Java-Klasse definiert: http://pmd.sourceforge.net/pmd-5.1.2/xref/net/sourceforge/pmd/lang/java/rule/design/GodClassRule.html

- ATFD (Access To Foreign Data) ist die Anzahl der direkten Zugriffe einer Klasse auf Attribute anderer Klassen. Diese darf nach der Regel „GodClass“ nicht höher als 5 sein.
- TCC (Tight Class Cohesion) ist für eine Klasse als Quotient definiert, die Anzahl an Methodenpaaren, die auf mindestens ein gemeinsames Attribut oder eine lokale Methode (dieser Klasse) zugreifen, die also eng miteinander verbunden sind, durch die Gesamtzahl alle möglichen Beziehungen zwischen den Methoden [LMD06]. Entsprechend kann TCC Werte zwischen 0 und 1 annehmen. Der berechnete Wert darf nach der Regel „GodClass“ 0,33 nicht unterschreiten. Ansonsten liegt die gewünschte Kohäsion der Methoden nicht vor und man kann die Klasse relativ leicht aufteilen.

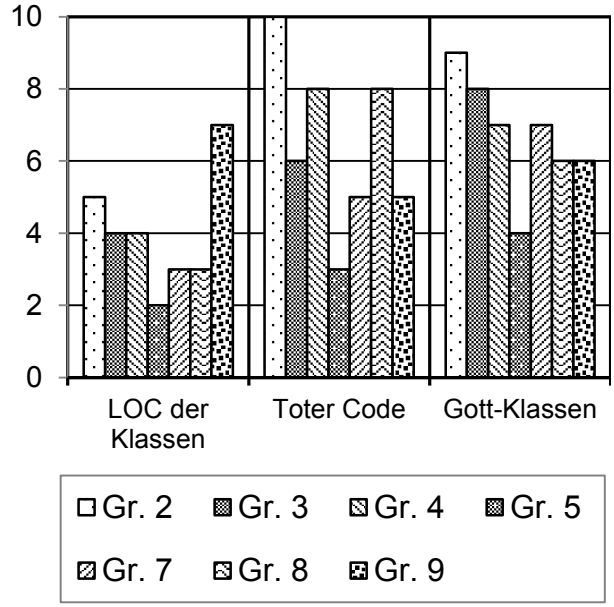


Abb. 6: Klassen

Wenn eines der Kriterien verletzt ist, wird eine Gott-Klasse vermutet. Bei uns waren meist zwei oder alle Kriterien betroffen. Die längste Klasse „GameController“ mit 992 LOC ist auch eine mögliche Gott-Klasse.

Schaut man sich die Projekte im Detail an, stellt man fest, dass es immer die gleichen Klassen sind, die als mögliche Gott-Klassen erkannt werden. Vier von sieben Gruppen haben nur eine Klasse für das Einlesen der Spielsituation in dem vorgegebenen Format angelegt, die den Inhalt der Datei einliest und abhängig vom gelesenen String viele unterschiedliche Objekte erzeugt. Oft hat diese Klasse dann sogar nur eine Methode wie die bereits als komplexeste Methode vorgestellte Methode „loadGame“ von Gruppe 4 (siehe Methoden).

Auch die Klassen, die jeweils den simulierten Computergegner realisieren, werden in fünf von sieben Gruppen als potentielle Gott-Klassen erkannt. In diesen Klassen wird die Spielsituation analysiert und ausprobiert, welcher Zug zu einer neuen, besonders vorteilhaften Spielsituation führt. Die Gruppe 5 hat für jede der geforderten KI-Stärken eine eigene Klasse angelegt, wobei alle drei als Gott-Klassen erkannt werden.

Außerdem sind die Klassen gefährdet, die den Zug des Menschen auf Zulässigkeit prüfen, alle möglichen Züge bestimmen und den Zug durchführen. Diese Klassen heißen „Game“, „GameController“, „Board“ oder „BoardController“, „SpielerController“ oder „Human“. Damit sind schon alle als Gott-Klassen erkannte Klassen genannt. Alle Gruppen haben sich mindestens eine Gott-Klasse geleistet (siehe Abb. 6).

Im Gegensatz zu den Control-Klassen sind bei fast allen Gruppen die Model-Klassen in Ordnung. Was nicht weiter erstaunlich ist, wenn man sich daran erinnert, dass es sich hier meist um einfache Datenhaltungsklassen handelt. Nur bei Gruppe 5, eine der Gruppen mit einem eher objektorientierten Ansatz, finden sich auch im Modell mögliche Gott-Klassen, z. B. die Klasse „Board“ mit WMC=112, ATFD=51, TCC=0.098.

Zusammenfassend lässt sich festhalten, dass bei uns spezielle Klassen dafür prädestiniert zu sein scheinen, sich zu Gott-Klassen zu entwickeln, nämlich diejenigen mit den algorithmisch anspruchsvollsten Aufgaben.

Auch wenn Bezeichner wie „…Manager“ oder „…Controller“ Hinweise auf potentielle Gott-Klassen liefern sollen, hat bei uns die Gruppe 5 die meisten potentiellen Gott-Klassen produziert, obwohl sie einen Ansatz mit mehr und umfangreicheren Model- als Control-Klassen gewählt hatte.

Auch das Muster, dass Gott-Klassen gern aus „entarteten“ Mediator-Klassen [Gamma u.a. 2004] entstehen, haben wir so bei uns nicht gefunden. Mediator-Klassen werden von den Studierenden häufig eingesetzt, um die Arbeit der verschiedenen Controller zu koordinieren. Sie heißen z. B. „MainController“. Bei uns scheint eher die algorithmische Komplexität der Auslöser für Gott-Klassen zu sein. Wir finden potentielle Gott-Klassen da, wo die algorithmisch anspruchsvolleren Aufgaben zu lösen sind. Es gelingt vielen Studierenden offenbar nicht, ein komplexes Problem in mehrere überschaubarere Teilprobleme zu zerlegen.

Toten Code haben wir in Form von unbenutzten Methoden, Parametern und Attributen gefunden. Es gab leere Methoden, die als Code-Rahmen aus dem UML-Modell generiert wurden, aber sich dann wohl doch als überflüssig herausgestellt haben. Wir haben auskommentierten Programmcode und sogar zwei unbenutzte Klassen gefunden. Die Projektlaufzeit ist mit drei Wochen zu knapp für den letzten Feinschliff oder die „Baustellen" geraten in Vergessenheit, weil sie ja beim Kompilieren keine Fehler melden und die Ausführung des Programms nicht stören.

Weitere Defekte

In den Spielprogrammen haben wir sehr häufig Zahlen in Bedingungen gefunden, z. B. für die maximale Anzahl der Mitspieler und die Anzahl der Teilschritte, aus denen ein Zug besteht. In einem Team, das sich wochenlang mit genau diesem Spiel beschäftigt hat, mag ein derartiger Ausdruck absolut verständlich erscheinen. Ohne sehr gutes Kontextwissen aus der Aufgabenstellung und den Spielregeln sind solche Code-Stellen nicht zu verstehen.

Ursache dafür scheint uns zu sein, dass gute Vorbilder fehlen. Vielmehr ist die Formulierung derartiger Bedingungen mit Literalen statt Konstanten in anderen Vorlesungen und Übungen schon aus Platzgründen üblich. Den Studierenden wird die Bedeutung von lesbarem Code erst dann klar, wenn sie in fremdem Code nach Fehlern suchen müssen.

Da Java die Anordnung der Komponenten einer Klasse nicht wirklich einschränkt, findet sich in manchen Klassen eine lustige Abfolge von Methoden und Variablen. Dass die Deklaration nicht wahllos erfolgen sollte, sondern insbesondere in umfangreichen Klassen einer vorgegebenen Ordnung entsprechen sollte, wird nicht vermittelt und erst beim Studium fremden Codes klar.

Diskussion der Ergebnisse

Wir haben relativ viele Mängel gefunden. Die Studierenden wussten allerdings beim Programmieren nicht, dass wir danach suchen würden.

Die Probleme bei der Namensgebung und die Verwendung von Literalen rühren wahrscheinlich daher, dass diese in den vorangehenden Vorlesungen und Übungen nicht thematisiert wurden. In den Vorlesungsfolien und beim Anschrieb der Übungsaufgaben werden aus Platz- und Zeitgründen besonders kurze Bezeichner verwendet. Man zeigt ja auch immer nur einen kleinen Ausschnitt. Die Studierenden sind somit überhaupt nicht vorbereitet auf die Notwendigkeit von aussagekräftigen Bezeichnern und die Vermeidung von Zahlen in Bedingungen zur Verbesserung der Wartbarkeit in umfangreicheren Projekten.

Der Zusammenhang zwischen der Länge einer Methode und ihrer Komplexität ist leicht nachzuvollziehen. Gruppen, die besonders viele lange Methoden geschrieben haben, hatten auch viele komplexe Methoden. Im Umkehrschluss kann man für die Studierenden die Empfehlung ableiten, keine langen Methoden zu schreiben. Lange Methoden sollten rechtzeitig aufgespalten werden. Derartiger Code ist auch besser zu testen.

Die Beseitigung von langen Parameterlisten durch das Refactoring „Introduce Parameter Object" ist durchaus kritisch zu sehen, da bei seiner Anwendung eine Klasse entsteht, die wiederum einen Konstruktor mit gleicher Parameterliste hat. Auch die Vermeidung von vielen Parametern durch Arraylisten oder Ähnlichem ist keine wirkliche Lösung, da dieser Code nicht besser lesbar ist. Vielmehr müssen von vornherein andere, kleinere Methoden entworfen werden, die Teilaufgaben lösen.

Die Länge einer Klasse scheint ein relativ leicht verständliches und leicht zu erkennendes Kriterium und ein gutes Indiz zu sein, um eine Klasse zu identifizieren, die Gefahr läuft, sich zu einer Gott-Klasse zu entwickeln. Die Regel für die Gott-Klassen mag für Anfänger schwierig zu durchschauen sein, aber „zu lang" kann wirklich jeder begreifen und im Auge behalten. Oft kann man beim Entwurf bereits erkennen, dass eine Klasse sehr viele Methoden bekommen soll. Dann könnten die Gruppenbetreuer rechtzeitig den Tipp geben, die Klasse zu zerlegen. Insbesondere, wenn ein Mangel an Kohäsion der Methoden vorliegt, sollte die Aufspaltung kein Problem sein.

Das Refactoring von einmal entstandenen Gott-Klassen ist relativ kompliziert. In dem Entwicklerforum „Stackoverflow" findet man auf die Frage nach dem Refactoring von Gott-Klassen als Antwort[6]: Das ist wie Jenga spielen. Wenn man an der falschen Stelle was wegnimmt, bricht alles zusammen.

[6] http://stackoverflow.com/questions/14870377/how-do-you-refactor-a-god-class

Das vorliegende Datenmaterial ist zu wenig umfangreich, als dass man vermutete Zusammenhänge, wie „Wenige Klassen führen zu mehr Gott-Klassen." oder „Mehr Control-Klassen führen zu mehr Gottklassen.", belegen könnte. In unseren Beispielen sind die Gottklassen in der Regel sehr lang.

Auch die interessante Frage, ob das MVC-Muster mit seiner Aufteilung in die Model- und die Control-Schicht die Entstehung von Gott-Klassen begünstigt, lässt sich (noch) nicht beantworten. Die Entwicklung eines Projekts gemäß MVC-Muster ohne Gott-Klassen wird angestrebt und sollte durchaus möglich sein.

Neben den Fragen danach, warum die Studierenden so viele Defekte in ihren Code einbauen und wie sie beseitigt oder vermieden werden können, stellt sich die Frage, wie die Studierenden auf die Kritik reagieren. Viele Studierende zeigen sich durchaus einsichtig und es ergeben sich konstruktive Diskussionen, wie man die Defekte hätte vermeiden können, z. B. zum Thema große Klassen oder schlechte Bezeichnerwahl. Daneben gibt es kritische Anmerkungen von den Studierenden zu den gewählten Regeln und Grenzwerten, z. B. zur maximalen Länge der Parameterliste und zu sehr kurzen Parameterbezeichnern. Uns ist natürlich klar, dass die Grenzwerte für die Messung relativ willkürlich gewählt sind und dass man darüber diskutieren kann. Grundsätzlich begrüßen wir auch diese Diskussionen über das Vorgehen, weil damit unser Ziel, Sensibilisierung für das Thema Codequalität, erreicht wird. Diskussionen über die Punktevergabe gibt es nicht. Sie scheint als fair empfunden zu werden.

Wie geht es weiter?

Die gewählten Grenzwerte haben sich bewährt und können beibehalten werden. Martin formuliert in [Martin 2008] sehr strenge Qualitätsmaßstäbe, denen selbst erfahrene Entwickler nur schwer genügen können. Da wir die Studierenden nicht völlig demotivieren wollen, haben bewusst erreichbare Ziele gesetzt. Mit den von uns gewählten Werten haben wir offenbar Grenzwerte gefunden, die die Studierenden in den meisten Fällen einhalten und nur selten verletzen.

Die Idee, bei der Bewertung von einem Guthaben von 10 Punkten bzw. 5 Punkten bei jedem Verstoß einen Punkt abzuziehen, halten wir weiterhin für didaktisch richtig. Da die Kategorien unabhängig voneinander betrachtet werden und es keine negativen Punkte gibt, haben selbst ganz viele Missgriffe in einer Kategorie, z. B. bei der Bezeichnerwahl, nicht den völligen Verlust aller Punkte zur Folge. In einer anderen Kategorie kann der Punkteverlust wieder ausgeglichen werden.

Im WS14/15 wollen wir Code-Qualität von Beginn an stärker thematisieren (siehe Vorgehensweise). Die Tutorials zu Clean Code und Refactoring sind fertig gestellt. PMD steht mit dem SoPra-Regelwerk als Eclipse-Plugin zur Verfügung. Wir hoffen, dass die Studierenden damit in der Lage sind, Mängel zu erkennen, und viele Mängel so beheben können.

Auch die Gruppenbetreuer sind aufgrund der im letzten SoPra durchgeführten Messungen und der Diskussion der Ergebnisse mehr für das Thema Code-Qualität sensibilisiert. Sie müssen den Gruppen dabei helfen, umfassendere Mängel wie potentielle Gott-Klassen oder zu lange Parameterlisten möglichst schon in der Entwurfsphase zu vermeiden, indem sie wie wir ein Gespür dafür entwickeln, wo sie entstehen könnten.

Zusammenfassung

Wir stellen vor, wie wir das Thema „Code-Qualität" in der Lehrveranstaltung Software-Praktikum eingeführt haben. Dabei gehen wir im Sinne des Blended Learning vor. Aus angeleiteten Lernprozessen und durch Selbststudium mit Hilfe von im Wiki bereitgestellten Unterlagen erarbeiten sich die Studierenden das Thema selbst. Wir haben für Anfänger angemessene Aspekte der Code-Qualität und geeignete Grenzwerte für die Messinstrumente festgelegt. Diese Vorbereitungen für ein erfolgreiches Selbststudium stellen wir hier vor. Durch die handlungsorientierte Vermittlung im Rahmen eines Software-Projekts soll den Studierenden die Bedeutung von Code-Qualität für die Software-Entwicklung leicht verständlich werden.

Durch das ständige Feedback des Tools erfahren die Studierenden, wie nahe sie dem Ziel von hoher Code-Qualität bereits gekommen sind und wo noch Mängel bestehen. Den Studierenden wird in den Tutorials und in einer Live-Demo von Eclipse gezeigt, wie sie welche Mängel leicht mit welchen Refactoringschritten beheben können und wo sie manuell eingreifen müssen. Durch die Diskussion mit den Dozenten und untereinander sollte den Studierenden die Bedeutung der Mängel klar werden.

Auf keinen Fall wollen wir die Bewertung zu strikt durchführen, beispielsweise die Vergabe des Scheins von der Einhaltung der Qualitätsgrenzen abhängig machen, da wir die Gefahr von adaptivem Verhalten sehen und verhindern wollen. Der Inhalt, die Funktionalität des Programms, ist immer noch wichtiger als die Form.

Literatur

Fowler, M. (1999): Refactoring: Improving the Design of Existing Code. Addison-Wesley, Reading, MA.

Gamma, Erich; Helm, Richard; Johnson, Ralph E.; Vlissides, John (2004): *Entwurfsmuster*. Elemente wiederverwendbarer objektorientierter Software. Addison Wesley, München.

Lanza, Michele; Marinescu, Radu; Ducasse, Stephan (2006): Object-oriented metrics in practice, Springer.

Martin, R. C. (2008): Clean code: a handbook of agile software craftsmanship. Pearson Education.

McCabe, Thomas (1976): A complexity measure. In: IEEE Transaction on Software Engineering, Nr. 4, S. 308-320.

Remmers, J.R. (2014): Clean Code – Code-Qualität im Software-Praktikum. BA-Arbeit, Fakultät für Informatik, TU Dortmund.

Riel, Arthur J. (1996): Object-oriented design heuristics. Addison-Wesley Publishing Company.

Schmedding, D. (2011): Teamentwicklung in studentischen Projekten. Software Engineering im Unterricht der Hochschulen (SEUH) 2011, München.

Spillner, Andreas; Linz, Tilo (2005): Basiswissen Softwaretest. dpunkt.verlag, Heidelberg.

Anhang A: Bewertungsschema

Kriterium	**Überprüfung durch**	**Bewertung**
Einhaltung der Java-Konventionen bei der Namensgebung	PMD	Max. 10 Punkte, pro Verstoß 1 Punkt Abzug
Sinnvolle Bezeichner	Stichproben, manuell	Max. 10 Punkte, pro Verstoß 1 Punkt Abzug
Zeilenlänge der Methoden <= 40	PMD	Max. 10 Punkte, pro zu lange Methode 1 Punkt Abzug
Parameteranzahl der Methoden <= 4	PMD	Max. 10 Punkte, pro Methode mit zu vielen Parametern 1 Punkt Abzug
Zyklomatische Komplexität der Methoden <= 10	PMD	Max. 10 Punkte, pro Verstoß 1 Punkt Abzug
Zeilenlänge der Klassen <= 400	PMD	Max. **5 Punkte**., pro zu lange Klasse 1 Punkt Abzug
Toter Code	PMD	Max. 10 Punkte, pro Verstoß ein Punkt Abzug
Gott-Klasse	PMD	Max. 10 Punkte, pro gefundene Gott-Klasse 1 Pkt. Abzug

Anhang B: SoPra-spezifische Regelmenge

```
<?xml version="1.0"?>
<ruleset name="SoPra"
        xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0 http://pmd.source-
forge.net/ruleset_2_0_0.xsd">
        <description>
                Diese Regelmenge wurde zur Nutzung in der Lehrveranstaltung "SoPra" der TU Dort-
mund erstellt.
                Sie besteht ausschließlich aus Standardregeln.
        </description>
<!-- Namensgebung-->
        <rule ref="rulesets/java/naming.xml/ShortVariable"/>
        <rule ref="rulesets/java/naming.xml/ShortMethodName"/>
        <rule ref="rulesets/java/naming.xml/ShortClassName"/>
        <rule ref="rulesets/java/naming.xml/VariableNamingConventions"/>
        <rule ref="rulesets/java/naming.xml/MethodNamingConventions"/>
        <rule ref="rulesets/java/naming.xml/ClassNamingConventions"/>
        <rule ref="rulesets/java/naming.xml/PackageCase"/>
        <rule ref="rulesets/java/naming.xml/AvoidDollarSigns"/>
        <rule ref="rulesets/java/naming.xml/SuspiciousConstantFieldName"/>
        <rule ref="rulesets/java/controversial.xml/AvoidLiteralsInIfCondition"/>
<!-- Methoden-->
        <rule ref="rulesets/java/codesize.xml/CyclomaticComplexity"/>
        <rule ref="rulesets/java/codesize.xml/ExcessiveMethodLength">
                <properties>
                        <property name="minimum" value="40"/>
                </properties>
        </rule>
        <rule ref="rulesets/java/design.xml/AvoidDeeplyNestedIfStmts"/>
        <rule ref="rulesets/java/codesize.xml/ExcessiveParameterList">
                <properties>
                        <property name="minimum" value="4"/>
                </properties>
        </rule>
<!-- Klassen-->
        <rule ref="rulesets/java/design.xml/FieldDeclarationsShouldBeAtStartOfClass"/>
        <rule ref="rulesets/java/codesize.xml/ExcessiveClassLength">
                <properties>
                        <property name="minimum" value="400"/>
                </properties>
        </rule>
        <rule ref="rulesets/java/design.xml/GodClass"/>
<!-- Unused Code-->
        <rule ref="rulesets/java/unusedcode.xml/UnusedPrivateField"/>
        <rule ref="rulesets/java/unusedcode.xml/UnusedLocalVariable"/>
        <rule ref="rulesets/java/unusedcode.xml/UnusedPrivateMethod"/>
        <rule ref="rulesets/java/unusedcode.xml/UnusedFormalParameter"/>
</ruleset>
```

Java, LEDs und ein RaspberryPi: Ein Projektversuch mit Erstsemestern

Axel Langhoff, Michael Striewe, Michael Goedicke
Universität Duisburg-Essen
{axel.langhoff,michael.striewe,michael.goedicke}@paluno.uni-due.de

Zusammenfassung

Bei der Einführung von Studierenden in die Programmierung stellt sich immer wieder das Problem, den Stoff attraktiv aufzubereiten und den Studierenden eine Möglichkeit zu geben, ihr Wissen auf eine motivierende Art anzuwenden. Für Studierende ohne Vorkenntnisse ist die Einstiegshürde in eine vollwertige Programmiersprache hoch. Hinzu kommt die Schwierigkeit mit dem frisch erworbenen Wissen eigene Ideen in ansehnliche Ergebnisse umzusetzen. Für Studierende mit großem Vorwissen hingegen besteht die realistische Gefahr sich in diesen Veranstaltungen zu langweilen.

In dem hier vorgestellten Ansatz haben wir versucht, unseren Studierenden in Form eines LED-Boards eine Umgebung zur Verfügung zu stellen, die (1) im Sinne einer Lernumgebung möglichst einfach zu verwenden ist, (2) dem unterschiedlichen und im Verlauf der Vorlesung wachsenden Kenntnisstand der Studierenden gerecht wird, (3) sichtbare und schöne Ergebnisse der Programmieranstregungen der Studierenden ermöglicht, (4) der Kreativität der Studierenden möglichst wenig Grenzen setzt, und (5) die Motivation der Studierenden steigert. Diesen Zielen liegt die Annahme zugrunde, dass eine motivierte Beschäftigung mit einem ansprechenden Projekt den Studierenden hilft, ihre Programmierkenntnisse zu vertiefen und zu festigen.

Neben dem LED-Board selbst berichten wir in diesem Beitrag über unsere Erfahrungen mit der Verwendung des LED-Boards in unserer Programmierungsvorlesung, sowie die Akzeptanz und den Lernerfolg unter den Studierenden.

1 Das LED-Board

Das von uns entwickelte LED-Board besteht ausschließlich aus Komponenten, die im regulären Elektronikhandel problemlos erhältlich sind und ohne besondere handwerkliche Qualifikation in der passenden Form zusammengebaut werden können. Die beiden wesentlichen Bestandteile sind abschneidbare LED-Bänder und ein RaspberryPi[1], um die LEDs auf den Bändern anzusteuern. Da die LEDs auf den Bändern einen Abstand von etwa 3,5 cm haben, haben wir uns für unsere Installation für ein Feld von 12x12 LEDs entschieden, das damit eine Fläche von etwa 40 x 40 cm einnimmt. Als Träger für die Montage der LEDs dient eine handelsübliche Blech-Lochplatte aus dem Baumarkt, so dass die Kabelführung bequem auf der Rückseite des Boards erfolgen kann. Jeder LED kann unabhängig von den Farben der anderen LEDs ein individueller RGB-Farbwert zugewiesen werden, so dass sich ein großes und leuchtkräftiges 144-Pixel-Farbdisplay ergibt (siehe Abbildung 1). Nach anfänglichen technischen Schwierigkeiten beim Zusammenbau, die teilweise auf mangelnde Dokumentation der verwendeten Komponenten zurückzuführen sind, konnten wir nach dem ersten Prototypen ein zweites, baugleiches Board an einem Arbeitstag zusammenbauen.

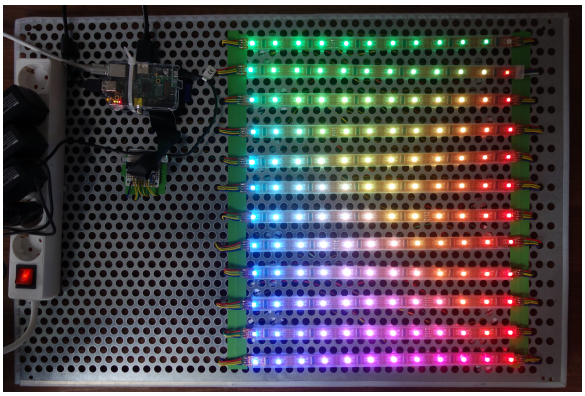


Abbildung 1: LED-Board

Um die direkte Ansteuerung der LEDs über die GPIO-Pins des RaspberryPi vor den Studierenden zu verbergen und die Ansteuerung der LEDs möglichst einfach zu gestalten haben wir eine Java-Bibliothek implementiert. Die Wahl fiel auf Java, da auch unsere Vorlesung für die das Board entwickelt wurde auf Ja-

[1] http://www.raspberrypi.org/

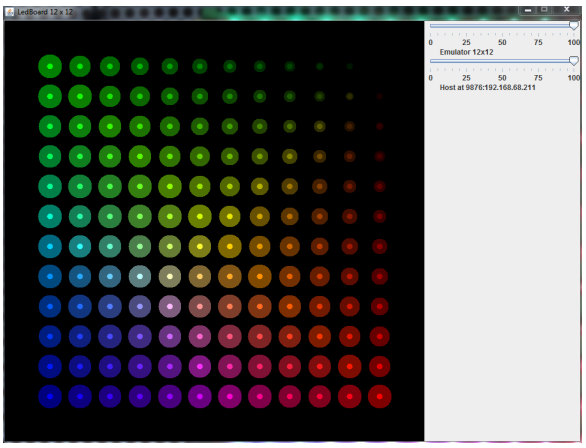


Abbildung 2: Emulator zum LED-Board. Die Größe der Kreise korrespondiert mit der Helligkeit der LEDs auf dem physischen Board. Mit den Schiebereglern rechts kann die Helligkeit des Emulators und des angeschlossenen Boards kontrolliert werden.

va aufbaut. Diese Bibliothek bietet mehrere leicht verständliche Methoden, um die einzelnen LEDs über ihre Koordinaten auf dem Board anzusteuern. Außerdem ermöglicht die Bibliothek den Studierenden am heimischen PC Programme für das Board zu entwickeln und die Ergebnisse unabhängig vom Board in einem Emulator zu sehen und zu testen (siehe Abbildung 2). Neben der Steuerung und Emulation des Boards wird von der Bibliothek auch ein einfaches Interface zur Verfügung gestellt, um Tastatureingaben auszuwerten und so die Entwicklung interaktiver Programme zu ermöglichen. Dieses Interface, wie auch der Rest der Bibliothek, verbergen außerdem das notwendige Threadhandling vollständig vor dem Benutzer.

Die Bibliothek stellt keine weiteren Anforderungen an die Umgebung und kann somit ohne Umstände durch Eintragung in den Java-Classpath in ein neues Java-Projekt eingebunden werden. Zu der Bibliothek wurden eine kleine, an Anfänger gerichtete Anleitung sowie mehrere Beispielprogramme angeboten, um die Verwendung der Bibliothek zu demonstrieren.

2 Einsatzziele des Boards in der Lehre

Ziel der Entwicklung des LED-Boards war es, eine Lernumgebung zu schaffen, die insbesondere in Lernveranstaltungen mit Erstsemestern eingesetzt werden kann. Das bedeutet, dass sowohl Studierende ohne Vorkenntnisse, als auch Studierende mit Programmiererfahrung aus Schule oder Freizeit angesprochen werden sollten, und dass das Board mit dem fortschreitenden Kenntnisstand im Laufe der Vorlesung mithalten sollte. Insbesondere war angestrebt, mit dem Board eine freie Umgebung anzubieten, in der die Studierenden keine vorgegebenen Aufgaben zu lösen haben, sondern eigene Projekte realisieren können. Auf diese Weise erhofften wir drei Ziele zu erreichen:

- Steigerung der Motivation der Studierenden aufgrund von eigener kreativer Tätigkeit,
- Vertiefung der vorhandenen Programmierkenntnisse durch praktische Anwendung,
- Sensibilisierung für Themen, die über den Stoff der Vorlesung bzw. die vorhandenen Kenntnisse hinaus gehen und die sich aus den selbstgewählten Zielsetzungen ergeben.

Der letzte Punkt ist insbesondere wichtig, da er sowohl unerfahrene als auch erfahrene Studierende anspricht: Unerfahrene Studierende können an ihren eigenen Projekten feststellen, wofür sie die Inhalte der Vorlesung benötigen, während erfahrene Studierende erkennen können, dass auch ihre Fähigkeiten noch nicht umfassend sind. Beide Gruppen werden somit zum Lernen motiviert.

2.1 Erforderliche und trainierte Fähigkeiten

Die einfachste Verwendung des LED-Boards besteht in der Darstellung von Standbildern. Die Farbwerte der LEDs können dazu theoretisch mit einem Befehl pro LED „per Hand“ gesetzt werden. Einzig ein sehr rudimentäres Verständnis von Arrays ist hierzu erforderlich und die Ergebnisse wären weder besonders interessant, noch lehrreich. Eine Abfolge von unzusammenhängenden Bildern wäre als weitere Verwendung denkbar, die erbrachte Programmierleistung ginge aber nicht über eine reine Fleißarbeit hinaus. Als strukturierendes Merkmal des Codes drängen sich in diesem Fall lediglich einzelne Methoden auf, die jeweils ein Einzelbild zeichnen und nacheinander aufgerufen werden. Die Inhalte einer typischen Anfängervorlesung zur objekt-orientierten Programmierung werden mit derartigen Projekte nur zum Teil abgedeckt. Eine umfassende Festigung und Vertiefung des gesamten Stoffes kann damit also nicht erreicht werden. Andererseits bietet die geringe Menge von notwendigen Kenntnissen auch schwächeren Studierenden die Chance, einen sichtbaren Erfolg zu erzielen.

Höhere Anforderungen stellen Animationen dar, die nicht aus einer Abfolge „von Hand“ gezeichneter Einzelbilder bestehen, sondern die durch eine dynamische Erstellung bzw. Veränderung der Einzelbilder erfolgen. Eine solche Animation erfordert mindestens das geschickte Zusammenspiel von Arrays und Schleifen. Als strukturierendes Merkmal des Codes bieten sich hier Klassen an, die jeweils einen Teil der Animation (z.B. ein veränderliches grafisches Objekt) realisieren und geeignete Methoden bereitstellen, um Instanzen zu manipulieren (z.B. die Position des grafischen Objekts zu ändern) und auf dem Board anzuzeigen. Projekte dieser Art decken damit bereits einen wesentlichen Teil einer typischen Anfängervorlesung

zur objekt-orientierten Programmierung ab und geben insbesondere anschauliche Beispiele für die dort vermittelten Kernkonzepte.

Mit Hinzunahme der Tastaturunterstützung sind auch interaktive Programme, insbesondere Spiele denkbar. Die Reaktion auf Benutzereingaben erhöht nochmal die Anforderung an die Programmstrukturierung, vor allem wenn es sich um kontinuierlich entgegengenommene Eingaben handelt bzw. das Programm im Hintergrund unabhängig von einer Eingabe fortschreiten soll. Da die Studierenden zur Verarbeitung von Benutzereingaben auch Keyevents aus den `awt`-Bibliotheken von Java verarbeiten müssen, entsteht auch ein erster leichter Anknüpfungspunkt zur selbstständigen Arbeit mit API-Dokumentationen. Insbesondere gehen solche Projekte zumindest punktuell deutlich über den Umfang einer einer typischen Anfängervorlesung zur objekt-orientierten Programmierung hinaus. Studierende, die solche Projekte gestalten zeigen damit eine deutliche Motivation, mehr als nur den notwendigen Stoff zum Bestehen der Klausur zu lernen.

2.2 Abgrenzung zu anderen (Lern-)umgebungen

Von Lernumgebungen wie Greenfoot (Henriksen u. Kölling, 2004) unterscheidet sich das LED-Board dadurch, dass es mit seiner geringen Zahl an farbigen LEDs zwar grundsätzlich weniger „hübsche“ Möglichkeiten der Ausgabe bietet, den Nutzern insgesamt jedoch mehr Freiheiten in der Anwendung bietet. Insbesondere macht es keine Vorgaben über ein Welt-Modell wie beispielsweise Kara[2] oder das Java-Hamster-Modell[3] und lässt damit auch anderen kreativen Ideen größeren Raum. Die bedeutendsten limitierenden Faktoren sind die Leistungsfähigkeit des RaspberryPis, die bei schnellen Bildwechseln zu Ruckeln führen kann, sowie die geringe Zahl an LEDs, die keine detaillierten Bilder erlaubt. Letzteres kann jedoch auch als Vorteil angesehen werden, da sich die Studierenden so auf algorithmisch und strukturell anspruchsvolle Projekte konzentrieren können, und weniger Mühe in optische Kosmetik investieren.

Gegenüber der reinen Programmierung mit Ausgabe auf der Kommandozeile (Java, C++, ...) sowie der Verwendung GUI-orientierter Sprachen wie VisualBasic oder Delphi bietet das LED-Board den Vorteil, dass sich schnell vergleichsweise spektakuläre Effekte erzielen lassen, ohne sich dazu in APIs einzuarbeiten, die den Rahmen einer Einführungsvorlesung üblicherweise sprengen bzw. Zeit für andere grundlegende Themen kosten. Dabei kommt auch zum Tragen, dass ein physisches Board mit leuchtkräftigen LEDs im Hörsaal bzw. Übungsraum einen anderen optischen Eindruck erzeugt als die Beamerprojektion einer Bildschirmausgabe. In wie weit sich dieser Effekt abnutzt, konnte im Rahmen unserer bisherigen Erfahrungen nicht untersucht werden.

Als Nachteil des LED-Boards muss angesehen werden, dass es mit Kosten für die Hardware verbunden ist und nicht in beliebiger Stückzahl vorgehalten werden kann. Andererseits ist dies aufgrund des verfügbaren Emulators auch nicht notwendig.

3 Einbettung in unsere Lehrveranstaltung

Das LED-Board wurde bisher im Wintersemester 2013/14 und im Sommersemester 2014 jeweils in der Erstsemestervorlesung zur Programmierung an der Universität Duisburg-Essen am Campus Essen eingesetzt. Der allgemeine Aufbau der Vorlesung sieht vor, dass die Studierenden neben 4 SWS Vorlesung und 2 SWS Globalübung im Semesterverlauf sechs Testate ablegen, in denen Programmieraufgaben zu bearbeiten sind und bei denen mindestens 400 von maximal möglichen 600 Punkten erworben werden müssen, um die Zulassung zur abschließenden Klausur zu erwerben. Das LED-Board wurde in diesem Konzept ergänzt, indem die Studierenden durch die erfolgreiche Bewältigung eines Programmierprojekts mit dem Board ebenfalls 100 Punkte erwerben konnten. Ein Projekt konnte dabei sowohl von einzelnen Studierenden, als auch von einer Gruppe aus bis zu drei Studierenden bearbeitet werden. Zusätzlich zu dieser formalen Einbindung des LED-Boards in die Vorlesung wurde ein „Publikumspreis“ ausgelobt, indem an einem Vorlesungstermin am Ende des Semesters alle teilnehmenden Studierenden ihr Projekt vorstellen und mittels Applaus durch ihre Kommilitonen bewerten lassen konnten. Für den ersten Platz wurde dann ein Sachpreis vergeben.

Die erfolgreiche Projektteilnahme im formalen Sinne schloss neben der Teilnahme an dieser Präsentation zur Demonstration eines lauffähigen Programms zudem die Teilnahme an einem anschließenden Codereview ein. Dieses Codereview sollte einerseits sicherstellen, dass die Studierenden ihren Code wirklich selbst erarbeitet hatten und bot andererseits eine Möglichkeit für die Lehrenden, den Teilnehmern Feedback zu ihrem Code und der Struktur ihrer Programme zu geben.

Die oben beschriebenen Teilnahmebedingungen, das LED-Board sowie die Java-Bibliothek wurden jeweils gegen Mitte des Semesters in der Vorlesung vorgestellt und den Studierenden zur Verfügung gestellt. Zu diesem Zeitpunkt konnten wir ein Minimalverständnis für die Programmierung und die in unserer Veranstaltung verwendete Entwicklungsumgebung Eclipse voraussetzen. Gleichzeitig ergab sich dadurch eine Bearbeitungszeit für das Projekt von etwa sechs Wochen, wobei die Studierenden diesen Zeitrahmen durch einen späteren Start oder eine frühere Abgabe selbstständig verringern konnten.

[2] `http://www.swisseduc.ch/informatik/karatojava/greenfootkara/index.html`

[3] `http://www.java-hamster-modell.de/index2.html`

Um einen hinreichenden Anspruch der Projekte zu gewährleisten, mussten alle Projekt-Ideen vorab mit dem Projektbetreuer abgesprochen und von diesem genehmigt werden. Die Schwelle ausreichender Komplexität wurde allerdings bewusst sehr niedrig gehalten, um auch die schwächeren Studierenden ansprechen zu können. Einzige feste Vorgabe in beiden Semestern war daher, dass ein bewegtes Bild auf dem LED-Board erzeugt werden musste. Aufgrund der Erfahrungen aus der ersten Durchführung haben wir uns beim zweiten Mal zusätzlich vorbehalten Projekte abzulehnen, die in dieser oder ähnlichen Formen schon von anderen Studierenden angemeldet worden waren, da sonst die Vorstellung am Ende des Semesters unter zu vielen ähnlichen Projekten gelitten hätte. Wurden Vorschläge aufgrund dieser Kriterien abgelehnt, wurden vom Projektbetreuer in der Regel Anregungen gegeben, wie die Projektidee geeignet abgewandelt werden konnte, um die Zulassungskriterien zu erfüllen. Wir haben außerdem eine Liste von möglichen Projekten bereit gehalten, falls Studierende gerne an dem Projekt teilnehmen wollten, aber noch keine eigenen Ideen hatten. Von diesem Angebot habe einige wenige Gruppen Gebrauch gemacht.

Um die Studierenden nicht darauf zu beschränken, ihre Programme am Emulator zu testen, wurde das Board auch für „Live-Experimente“ im Rahmen ohnehin mehrfach pro Woche stattfindender Tutorien zur Verfügung gestellt. In diesen Tutorien standen schon vorher studentische Tutoren dafür bereit, Studierende individuell bei der Arbeit an Übungsaufgaben zu betreuen, so dass der einzige Mehraufwand darin bestand, diese studentischen Tutoren in der Handhabung des LED-Boards zu schulen. Durch dieses Angebot konnte insbesondere sichergestellt werden, dass die Studierenden eine Möglichkeit hatten, sich rechtzeitig mit Performanceunterschieden zwischen ihrem eigenen PC und dem RASPBERRYPI vertraut zu machen und tutorielles Feedback zu ihrem Projekt zu erhalten. Außerdem wurde so eine Umgebung geschaffen in der Studierende sich gegenseitig bei der Arbeit am Projekt beobachten konnten und auf diese Weise ins Gespräch kamen, um wechselseitig weiteres Feedback zum Projekt zu geben und zu erhalten.

4 Teilnehmerzahlen und Projekte

Im Wintersemester 2013/2014 nahmen zum Zeitpunkt der Vorstellung des LED-Boards knapp 300 Studierende ernsthaft an der Vorlesung teil, von denen 29 (verteilt auf 18 Gruppen zu einer bis drei Personen) erfolgreich ein Projekt mit dem LED-Board durchführten. Im Sommersemester 2014 führten von etwa gleich vielen Vorlesungsteilnehmern 48 Studierende in 24 Gruppen erfolgreich ein Projekt mit dem LED-Board durch (siehe Tabelle 1). Hinzu kamen in beiden Semestern weitere Anmeldungen, für die bis zum Einsendeschluss jedoch gar keine oder keine zufriedenstellende Einreichung abgegeben wurden. Die

	WS 2013/14	SS 2014
Einzelteilnehmer	10	7
2er-Gruppen	5	10
3er-Gruppen	3	7
Gesamtzahl Teilnehmer	29	48

Tabelle 1: Teilnehmerzahlen in den beiden Projektdurchläufen im Wintersemester 2013/14 und Sommersemester 2014.

geringere Teilnehmerzahl im Wintersemester dürfte auf die spätere Vorstellung des LED-Boards zurückzuführen sein. Das Board befand sich zu dieser Zeit noch in der Entwicklung und vor allem die Java-Bibliothek erhielt in diesem Semester noch mehrere Updates.

Unter den in beiden Semestern eingereichten Projekten fand sich eine sowohl in Art und Idee, als auch programmiererischer Komplexität und Schwierigkeit breite Auswahl an Programmen. Von der nichtinteraktiven Anzeige einfacher Flaggen anlässlich der 2014 stattfindenden Fußball-WM, die mit verschiedenen Animationen ineinander übergeblendet wurden, über liebevoll animierte Landschaften, Feuerwerke, interaktive Laufschriften (eine davon über Twitter steuerbar!) und Spiele wie *Snake*, *4 gewinnt*, *2048* und *Tetris*, bis zu einem ganzen Programmierungsframework für das Board, dessen Fähigkeiten seine Programmierer durch mehrere verhältnismäßig aufwändige Spiele demonstrierten, war deutlich sichtbar eine weite Spanne von Programmierungsanfängern bis zu bereits erfahrenen Programmierern abgedeckt.

Es kann damit bereits an dieser Stelle festgehalten werden, dass das Ziel, die Kreativität der Studierenden nicht einzuschränken und gleichzeitig sowohl Anfänger als auch Fortgeschrittene anzusprechen, voll erreicht wurde. Im Folgenden werden einige Aspekte zum Umgang mit dem Board allgemein und an exemplarischen Projekten aus beiden Semestern im speziellen diskutiert.

4.1 Beispiel 1: Einfache Animation (SS 2014)

Beim ersten hier vorgestellten Projekt handelt es sich um eine einfache Animation. Ein Ballon bewegt sich vor vorbeiziehenden Wolken und einem statischen Hintergrund langsam hin und her (siehe Abbildung 3). Das Projekt erfüllt damit die Minimalanforderungen, ohne in seinem Anspruch an irgendeiner Stelle darüber hinaus zu gehen.

Die Umsetzung erfordert einen sicheren Umgang mit Arrays und Schleifen, ein Verständnis für die korrekte Zeichenreihenfolge bei sich verdeckenden grafischen Objekten und bietet Möglichkeiten, einen strukturierten Programmaufbau zu üben.

Im uns vorliegenden studentischen Quellcode lässt sich tatsächlich eine klare Strukturierung erkennen. Der Code ist vollständig in einer Javaklasse untergebracht und durch Methoden in sinnvolle Einheiten

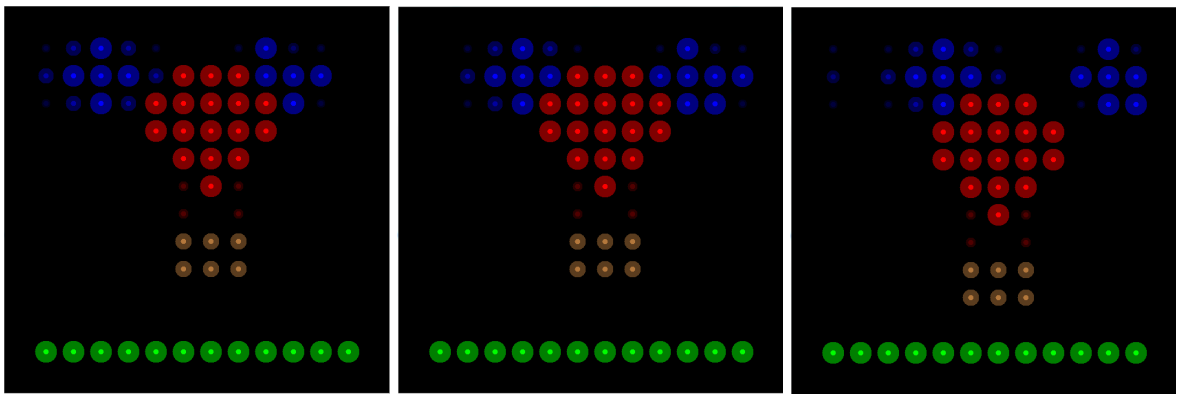


Abbildung 3: Drei direkt aufeinander folgende Screenshots aus einer einfachen Animation mit einem bewegten Ballon vor vorbeiziehenden Wolken, siehe Abschnitt 4.1.

gegliedert. Auch wenn mehr Strukturierung denkbar wäre, ist der verwendete Ansatz für das Problem völlig ausreichend und ein Grundverständnis für die Programmierung deutlich aus dem Quellcode abzulesen. Weitergehende Konzepte der Objektorientierung, die auch Thema der Vorlesung waren, fanden sich in diesem Projekt nicht wieder.

Projektumfang und Programmaufbau legen die Vermutung nahe, dass es sich bei den Studierenden um Anfänger im Bereich der Programmierung gehandelt hat, die mit diesem Projekt Sicherheit erhalten wollten, indem sie in der Vorlesung erworbene grundlegende Programmierfähigkeiten durch weitere Übung festigen wollten. Insbesondere ist nicht zu erkennen, dass die Studierenden Kenntnisse von außerhalb der Vorlesung in das Projekt eingebracht haben oder sich für das Projekt zusätzliche Kenntnisse angeeignet und ausprobiert haben.

4.2 Beispiel 2: Interaktives Spiel (WS 2013/2014)

Das zweite hier vorgestellte Projekt ist eine Implementierung des Spiels „Snake“. Bei diesem Spiel bewegt sich eine vom Spieler zu steuernde Schlange in den vier Grundrichtungen über das Spielfeld und sammelt Äpfel auf, wodurch die Schlange immer länger wird. Das Spiel ist beendet, sobald die Schlange entweder mit sich selbst oder dem Spielfeldrand kollidiert. Tatsächlich war die Spielidee so beliebt, dass allein im Wintersemester 2013/2014 vier verschiedene Teams Implementierungen dieses Spiels eingereicht hatten. Eine dieser Implementierungen ist beispielhaft in Abbildung 4 wiedergegeben.

Für eine gute Implementierung von Snake bietet sich der Einsatz von Objekten und Klassen an, insbesondere die Verwendung einer dynamischen Datenstruktur. Die Schlange besteht aus einer Folge von Einzelpositionen, was die Verwendung eines Arrays oder einer Liste zum Verwalten der Einzelpositionen nahelegt. Eine Fortbewegung der Schlange im Spielverlauf bedeutet aus Spielersicht, dass alle einzelnen Schlangenglieder um ein Feld vorwärts bewegt werden. Aus Sicht des Programmierers ist es effizienter das letzte Schlagenglied zu entfernen und ein neues Glied an den Kopf der Schlange zu setzen. Die mittleren Schlangenglieder müssen dadurch nicht geändert werden. Isst die Schlange einen Apfel, muss ein neues Schlangenglied an die Schlange angehängt werden.

Die hier beschriebenen Mechanismen sind mit einer Liste deutlich einfacher zu implementieren als mit einem Array.

Sowohl Objekte und Klassen, als auch dynamische Datenstrukturen wie Listen wurden in der Vorlesung rechtzeitig eingeführt, um die Verwendung in einem der studentischen Projekte zu ermöglichen. Außerdem erfordert eine Implementierung des Spiels eine einfache Gameloop, d.h. eine Schleife, die wiederholt Nutzereingaben abfragt und die Spielwelt entsprechend anpasst. Zu beachten ist dabei, dass das Spiel beim Warten auf Eingabe nicht pausiert, sondern sich der Spielzustand auch unabhängig von einer Eingabe weiterentwickelt.

Allen vier eingereichten Projekten war anzusehen, dass die Studierenden sich um eine geschickte Strukturierung ihres Codes bemüht hatten, was in einigen Fällen gut, in anderen weniger gut gelang. Die Projekte wurden in zwei bis zehn Klassen untergliedert und in einem Fall auf mehrere Packages verteilt. In einem Projekt erfolgte ein Rückgriff auf die in der Vorlesung vorgestellten Listenkonzepte.

In allen vier Projekten ist es den Studierenden gelungen, ein funktionierendes Snake-Spiel zu implementieren. Neben der Basisfunktionalität fanden sich in allen Projekten zusätzliche Gimmicks wie Eingangs- und Abschlussanimationen, wählbare Schwierigkeitsgrade und Punktzahlanzeigen, was die Motivation der beteiligten Studierenden unterstreicht.

In einem der Projekte ist die selbstständige Verwendung von Threads zu sehen, die anderen drei Projekte verwenden keine über den Inhalt der Vorlesung hinausgehenden Konzepte.

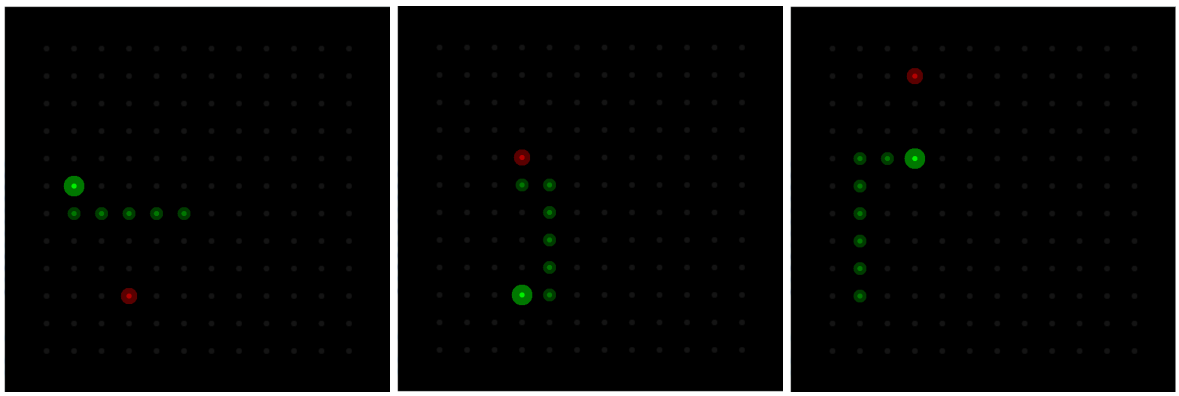


Abbildung 4: Drei Screenshots aus aufeinander folgenden Spielsituationen im Spiel „Snake“, siehe Abschnitt 4.2

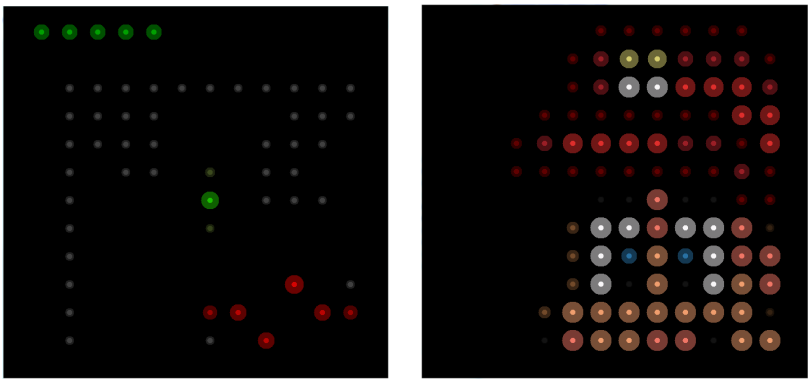


Abbildung 5: Ausgewählte Demonstrationen des studentischen Framework aus Abschnitt 4.3. Links: Ein Helikoptershooter mit dem Spieler in grün, beweglichen Hindernissen in rot und Spielumgebung in grau. Rechts: Supermario, eingelesen aus einer Bilddatei.

4.3 Beispiel 3: Framework (WS 2013/2014)

Das dritte hier vorgestellte Projekt hat unsere Erwartungen bei weitem übertroffen. Die dahinter stehenden zwei Studierenden haben sich nicht damit zufrieden gegeben ein Spiel zu schreiben, sondern haben gleich ein ganzes Framework für das LED-Board angefertigt und dieses Framework an mehreren nicht trivialen Anwendungen und Spielen demonstriert. In Abbildung 5 sind zwei ausgewählte Beispiele zu sehen. Im Framework ist unter anderem Unterstützung für das Verarbeiten von Polygonen und Sprites zu finden. Des weiteren findet sich eine Kollisionserkennung für diese Objekte, wahlweise per Hitbox oder pixelbasiert. Außerdem besteht die Möglichkeit, Bilder aus Bitmasken oder gespeicherten Dateien in üblichen Grafikformaten auf dem Board darzustellen. Dabei unterstützt das Framework auch das Scrolling von Bildern oder Masken, deren Auflösung größer als die des LED-Boards ist. Zur Demonstration des Frameworks waren mehrere Programme beigelegt, die das Framework nutzen und sehr gut strukturiert sind. Framework und Demonstrationsprogramme bestehen aus insgesamt 95 Klassen, die in 26 Packages organisiert sind.

Der hohe Organisationsgrad, die Komplexität und die Verwendung fortgeschrittener Konzepte wie Quadtrees zeigen, dass die beiden Studierenden schon vor unserer Programmierungsvorlesung umfangreiche praktische Erfahrungen gesammelt haben.

Der Aufwand für dieses Framework muss schon allein aufgrund des Umfangs sehr hoch gewesen sein und belegt, dass mit dem LED-Board auch fortgeschrittene Studierende angesprochen und motiviert werden können.

4.4 Allgemeines zu den Projekten

Die vorgestellten Projekte gewähren einen guten Eindruck von der Bandbreite der abgegebenen Projekte. Unter allen angenommenen und am Ende in der Vorlesung präsentierten Programmen fanden sich naturgemäß auch ein paar schwächere Projekte, deren Urheber vor allem zwecks Erwerb der Klausurzulassung teilnahmen. Der Anteil dieser Projekte an sämtlichen präsentierten Projekten blieb allerdings mit unter einem Fünftel erfreulich gering. Insgesamt ungefähr ein Viertel der Anmeldungen führte allerdings gar nicht erst zu Abgaben.

Aus den Erfahrungen der Tutorien, des Codereviews und des uns vorliegenden Quellcodes offenbarte sich an vielen Stellen eine große Experimentierfreude. So versuchten sich viele Studierende in der Verwendung von Konzepten, die bis dahin nicht Teil ihres Studi-

Projekt	LOC	Klassen
Einfache Animation (SS 2014)	118	1
Interaktives Spiel ((WS 2013/14)	302	3
Framework ((WS 2013/14)	4509	95
Durchschnitt aller Projekte	710	6

Tabelle 2: Umfang ausgewählter Projekte und Durchschnitt aller Projekte. Die LOC-Metrik berücksichtigt nur reine Codezeilen und ignoriert Leerzeilen und Kommentarzeilen.

ums waren. Vielfach wurden Threads verwendet, die JavaAPI wurde mehrfach für die Generierung von Zufallszahlen und für Filehandling verwendet und damit einhergehend war auch Exceptionhandling zu sehen. Ferner zeigten sich immer wieder Studierende offen für Vorschläge, die über ihr ursprüngliches Projektziel hinausgingen. Nicht immer waren diese Experimente von Erfolg gekrönt, dennoch waren sie ein weiteres Indiz für die Motivation. Umgekehrt scheiterten leider auch einige Studierende an ihren ursprünglichen Zielen und mussten sie herunterschrauben. In den Codereviews zeigte sich zudem, dass auch in den Teams stets alle Beteiligten tatsächlich programmiert hatten. Es liegt nahe, dass dies zumindest zum Teil auch darauf zurückzuführen ist, dass das Board keine Möglichkeiten bietet, sich mit anderen Arbeiten (z.B. dem Erstellen hübscher Icons) einzubringen.

Allen Projekten war zu eigen, dass den Teilnehmern die Notwendigkeit guter Codestrukturierung unmittelbar vor Augen geführt wurde, wenn auch nicht alle Studierenden in der Lage waren, eine solche Strukturierung mit ihren Mitteln zu erreichen. Einige Daten zum Umfang der oben vorgestellten Projekte gemessen in Codezeilen (LOC) sowie zum Organisationsgrad gemessen in der Zahl der Klassen sind in Tabelle 2 aufgeführt. Der in der Tabelle angegebene Durchschnitt bezieht sich auf alle abgegebenen Projekte aus beiden Semestern. Dabei ist zu bemerken, dass sehr starke Unterschiede zwischen einzelnen Projekten auftragen: Drei Projekte mit je über 1000 LOC waren jeweils in nur einer Klasse realisiert, während fünf andere Projekte jeweils einen Schnitt von weniger als 50 LOC pro Klasse erreichten.

In den Tutorien und spätestens im abschließenden Codereview konnten den Studierenden in den meisten Fällen zahlreiche Hinweise gegeben werden, die ihren Code verbessern und vor allem den Sinn vieler in der Vorlesung vorgestellten Konzepte an interessanten, da von den Studenten selbst ausgedachten und gewählten, Szenarien verdeutlichen. Softwarekonstruktion und wie man guten Code schreibt waren nur am Rande Thema unserer Programmierungsvorlesung. Durch das LED-Projekt konnten die meisten Teilnehmer dennoch in einem Maß für diese Problematiken sensibilisiert werden, das wir ohne das Projekt nicht erreicht hätten.

Testatpunkte	Spaß	Training	Preis
82%	66%	48%	21%

Tabelle 3: Motivation zur Teilnahme (Mehrfachnennungen möglich)

5 Rückmeldung der Studierenden

Neben persönlichem Feedback haben wir zum Abschluss der Projekte in beiden Semestern jeweils einen anonymen Umfragebogen zum LED-Board ausfüllen lassen. Gefragt wurden ausschließlich Studierende, die auch ihr Projekt vorgestellt hatten. Die hier präsentierten Ergebnisse wurden aus insgesamt 71 ausgefüllten Umfragebögen beider Semester zusammengetragen. Gefragt wurde

- nach den Gründen zur Teilnahme an dem LED-Projekt,
- ob sich das eigene Programmierverständnis nach eigener Einschätzung verbessert hat,
- ob die Teilnehmer für sich ein überdurchschnittliches Klausurergebnis erwarteten,
- wie sich die Motivation der Teilnehmer mit dem Projektverlauf entwickelt hat
- und ob das Projekt die Erwartungen der Teilnehmer erfüllt hat.

Außerdem bot der Umfragebogen Raum für freie Kommentare aller Art.

5.1 Gründe zur Projektteilnahme

Als mögliche Gründe für die Projektteilnahme waren

- Testatpunkte
- Spaß
- Training der eigenen Fähigkeiten
- der Sachpreis für den ersten Platz
- und sonstiges

gegeben, wobei Mehrfachnennungen möglich waren. Das zusammengefasste Ergebnis ist in Tabelle 3 angegeben. Insgesamt haben 82% der Teilnehmer den Erwerb von Testatpunkten als Grund genannt. Bemerkenswert ist dabei jedoch, dass viele Teilnehmer zum Zeitpunkt der Vorstellung ihrer Projekte ihre Klausurzulassung bereits hatten, und sie dennoch zu Vorstellung und Codereview gekommen sind.

Sehr erfreulich ist die hohe Zahl der Teilnehmer die Spaß am Board (66%) oder Training der eigenen Fähigkeiten (48%) als Motivation angegeben haben. Zu untersuchen bleibt, aus welchen Gründen sich Studierende gegen eine Teilnahme am LED-Projekt entschlossen haben.

hoch	fallend	niedrig	steigend
46%	21%	6%	27%

Tabelle 4: Motivationsverlauf

5.2 Verbesserung des eigenen Programmierverständnisses

Die Verbesserung ihres eigenen Programmierverständnisses sollten die Teilnehmer auf einer Skala von 0 (gar nicht) bis 4 (sehr stark) einschätzen. Über alle Teilnehmer gemittelt hat sich hier ein Wert von ca. 2,3 ergeben. Für den Großteil der Teilnehmer erbrachte das Projekt im subjektiven Eindruck also eine spürbare Verbesserung der eigenen Fähigkeiten.

Studierende, die bei der Motivation angegeben haben, ihre eigenen Fähigkeiten trainieren zu wollen, haben in der Kategorie „Verbesserung des eigenen Programmierverständnisses“ im Schnitt ca. 2,6 angegeben, was signifikant über dem Gesamtschnitt von 2,3 liegt.

5.3 Erwartungen an die Klausur

Mit den Antwortmöglichkeiten „Ja“ und „Nein“ sollten die Teilnehmer einschätzen, ob sie ein überdurchschnittliches Klausurergebnis erzielen werden. 45% erwarteten ein überdurchschnittliches Ergebnis, die restlichen 55% nicht. Da die Umfrage anonym durchgeführt wurde, kann leider nicht überprüft werden, ob die Erwartungen mit den tatsächlichen Klausurergebnissen übereinstimmen.

5.4 Motivationsverlauf

In der Frage nach dem Motivationsverlauf war eine der vier Möglichkeiten „hoch“, „fallend“, „niedrig“, „steigend“ zu wählen. Die Ergebnisse sind in Tabelle 4 aufgeführt.

73% der Studierenden gab einen dauerhaft hohen oder mit der Zeit steigenden Motivationsverlauf an. Interessant zu erfahren wäre für zukünftige Erhebungen, warum bei den übrigen 27% die Motivation gelitten hat. Es ist nicht ersichtlich, ob die Gründe dafür im LED-Projekt selbst oder in äußeren Umständen wie z.B. Zeitdruck liegen.

Eine weitere Analyse der Daten zeigt, dass Teilnehmer die aufgrund von Spaß am Board oder um die eigenen Fähigkeiten zu trainieren teilnahmen, leicht überdurchschnittlich oft eine hohe oder steigende Motivation aufwiesen. Der Preis hatte dagegen keinen erkennbaren Einfluss auf die Motivation. Teilnehmer die Testatpunkte als einen von mehreren Teilnahmegründen angegeben haben, wiesen im Schnitt einen geringfügig schlechteren Motivationsverlauf auf. Betrachtet man die Teilnehmer, die ausschließlich der Testatpunkte wegen ein Projekt eingereicht hatten, weist sogar nur ein Drittel dieser Teilnehmer eine hohe oder steigende Motivation auf.

5.5 Erwartungen an das Projekt

Auf einer Skala von 0 (gar nicht) bis 4 (vollständig) sollte angegeben werden, ob die Erwartungen der Teilnehmer an das Projekt erfüllt wurden. Mit dem Mittelwert von ca. 3,0 können wir erfreulicherweise feststellen, den Erwartungen im Wesentlichen gerecht geworden zu sein.

5.6 Freie Kommentare

Neben der Umfrage sprechen vor allem die Einzelkommentare, die direkten Rückmeldungen an die Projektbetreuer sowie die rege Teilnahme an dem Wettbewerb für eine große Akzeptanz des Projektes. Kritisiert wurden von den Studierenden insbesondere technische Aspekte wie z.B. die geringe Zahl von 12x12 LEDs und die Leistungsfähigkeit des RaspberryPis.

Die Umfrageauswertung lässt insgesamt den Schluss zu, dass das Board und der zugehörige Emulator gute Werkzeuge zum selbstständigen Training sind und von den Studierenden auch zu diesem Zweck akzeptiert werden.

6 Ausblick und weiterer Einsatz

In kommenden Programmierveranstaltungen werden wir sicherlich wieder einen Wettbewerb auf dem LED-Board anbieten. Da das LED-Board fertiggestellt ist und die Bibliothek von ihren Kinderkrankheiten befreit, ist der durch den Wettbewerb entstehende Mehraufwand für das Lehrpersonal sehr gering für den deutlich positiven Effekt.

Neben der gegenwärtigen Verwendung ist es auch denkbar, das Board oder zumindest den Emulator in die eigentliche Vorlesung und/oder die Übungen zu integrieren. Insbesondere Schleifen und Arrays lassen sich an dem Board sehr anschaulich demonstrieren. Für Übungsaufgaben bieten sich Board und Emulator ebenso an.

Mit kleinen Anpassungen der LED-Bibliothek ist auch eine Einbindung in unserer automatisches Übungs- und Prüfungssystem JACK (Striewe u. a., 2009) möglich. Sich auf automatisch auswertbare Aufgaben zu konzentrieren würde allerdings einen sehr großen Aufwand und eine starke Einschränkung der Nutzungsmöglichkeiten des Boards bedeuten.

Neben dem Einsatz in der Erstsemestervorlesung wurde das LED-Board auch schon mit interessierten Schülern in Workshops, fortlaufenden AGs oder anderen Veranstaltungen zur Studiengangswerbung in der Informatik verwendet.

7 Fazit

Insgesamt betrachten wir den Einsatz des LED-Boards als erfolgreich, da (1) das Board von den Studierenden wie erwartet einfach und ohne nennenswerte technische Probleme nutzbar war, (2) sowohl einfache als auch anspruchsvolle Projekte auf dem Board realisiert wurden, (3) optisch ansprechende Ergebnisse in einer wettbewerbsartigen Form im Hörsaal

präsentiert werden konnten, (4) eine große Breite an verschiedenartigen Projekten realisiert wurde und (5) die Mehrheit der teilnehmenden Studierenden eine hohe Motivation aufwies und durch ihre Arbeit belegte.

Dem Aufwand bei der Vorbereitung des Boards steht somit insbesondere ein messbarer Erfolg bei der Steigerung der Motivation der Studierenden entgegen, der mindestens mit einem subjektiv höheren Lernerfolg verbunden ist.

Literatur

[Henriksen u. Kölling 2004] HENRIKSEN, Poul ; KÖLLING, Michael: greenfoot: Combining Object Visualisation with Interaction. In: *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. New York, NY, USA : ACM, 2004. – ISBN 1–58113–833–4, S. 73–82

[Striewe u. a. 2009] STRIEWE, Michael ; BALZ, Moritz ; GOEDICKE, Michael: A Flexible and Modular Software Architecture for Computer Aided Assessments and Automated Marking. In: *Proceedings of the First International Conference on Computer Supported Education (CSEDU), 23 - 26 March 2009, Lisboa, Portugal* Bd. 2 INSTICC, 2009, S. 54–61

Lecture Engineering

Thomas Lehmann und Bettina Buth, HAW Hamburg
thomas.lehmann|bettina.buth@haw-hamburg.de

Zusammenfassung

Für die Überarbeitung des Moduls Software Engineering im Studiengang Mechatronik der HAW Hamburg wurde ein kompetenzorientierter Ansatz verwendet. Dabei wurden Anleihen an die Vorgehensweisen und Prinzipien des Softwareengineerings genutzt, um systematisch zunächst die Kompetenzen und dann daraus die Inhalte des Moduls zu entwickeln. In diesem Artikel berichten wir über die bisherigen Erfahrungen bei der Konzeption und der Durchführung des Moduls anhand eines Fallbeispiels.

Ausgangssituation

An der HAW Hamburg wird der Studiengang Mechatronik gemeinsam von den Bereichen Maschinenbau, Elektrotechnik, Informatik und Fahrzeugtechnik durchgeführt. Das Department Informatik betreut entsprechend den Informatik-Anteil des Studiengangs. Durch personelle Änderungen ging die Verantwortung für das Modul Software Engineering zum Wintersemester 2013/14 an uns über. Die Übernahme sollte dazu genutzt werden, es von Grund auf neu zu konzipieren. Die Mechatronik vereint Mechanik, Elektrotechnik und Informatik, sodass man hier Studierende mit einer geringeren Affinität zu Software antrifft als in den Informatik-Studiengängen. In diesem Zusammenhang scheint es sinnvoll, den angehenden Mechatronikern nicht Softwareengineering im engeren Sinne, sondern die Perspektive des systematischen Vorgehens aus System- und Softwaresicht nahe zu bringen. Eine direkte Übernahme des Moduls Software Engineering aus der (Technischen) Informatik war deshalb nicht angebracht, da dieses nur die Softwareseite beleuchtet und von anderen Vorkenntnissen ausgeht. Das Modul musste auf die anderen Vorkenntnisse in Programmieren und die andere Art von Studierenden in diesem Studiengang angepasst sein.

Das Modul kann aufbauen auf Vorkenntnisse aus zwei Semestern Programmierung in C/C++. Insgesamt entspricht der Kenntnisstand hier ca. 40-50% dessen was in den Informatik-Studiengängen der HAW in den ersten beiden Semestern vermittelt wird. Die planerischen Rahmenbedingungen des Fachs gehen von einer Vorlesung mit 3 Vorlesungsstunden und 1 Praktikumsstunde pro Woche aus, üblich erfolgt eine Blockung der Praktika in 4 Termine a 4 Stunden.

> "Wo der Überblick fehlt, ist das Grauen nicht weit."(Lotter (2005))

Mit der Entscheidung zur Neugestaltung ergab sich die Frage, ob es einen systematischen Ansatz zur fachlichen und didaktischen Ausgestaltung eines Moduls gibt? Aus der Diskussion innerhalb der Hochschule bietet sich hier ein kompetenzorientiertes Vorgehen an. Wie behalte ich den Überblick bei der Entwicklung des Moduls? Da es hier um eine Art Produktentwicklung geht, entstand die Idee, Ideen und Methoden des Softwareengineerings auf die Entwicklung des Moduls Software Engineering anzuwenden. Dazu wurden Äquivalenzen zwischen den Entwicklungsschritten im V-Modell und der Entwicklung eines Moduls identifiziert. Im Folgenden wird beschrieben welche Anleihen gemacht, wie diese im kompetenzorientiertem Umfeld eingesetzt und welche Erfahrungen dabei gesammelt wurden.

Erste Schritte

Das Requirements Engineering (vgl. z. B. Hammerschall u. Beneken (2013)) sammelt die Erwartungen und Anforderungen an ein Produkt in seinen Facetten ein und gibt Leitlinien zu dieser Anforderungserhebung. In einer Vision wird ein Leitbild für das Produkt beschrieben, welche durch die Stakeholder in seinem Kontext näher definiert wird. Die Anforderungen werden systematisch dokumentiert und in einen Entwurf umgesetzt. Mit diesen Kernideen sind wir in das Erstellen von Requirements für das Modul eingestiegen.

In einem Textdokument haben wir zunächst unsere Vorstellung über ein Softwareengineering für einen Mechatroniker, somit das Ziel des Moduls im Kontext des Curriculums, dokumentiert. Bei der Betrachtung der Stakeholder lag der Fokus auf dem Hochschulkontext, also Studierende, Professoren/Dozenten, Labormitarbeiter, Stundenplaner und Studiengangskoordinatoren. Externe Stakeholder wie Personalverantwortliche wurden im ersten Iterationsschritt der Anwendung dieses Vorgehens nicht betrachtet. Diese hätten wie bei der Studie zu erforderlichen Kompetenzen (vgl. Hummel (2013)) für die Auswahl der Kompetenzen mit hinzugezogen werden müssen. Hier wurde zum Ausprobieren der Vorgehensweise auf die eigenen Erfahrungen aus der Praxis zur Definition der Kompetenzen vertraut.

Die Hochschule gibt einen Rahmen für die Umsetzung vor. Dieser Rahmen aus Anzahl der Semesterwo-

chen, Semesterwochenstunden (3+1 SWS), Kohortengröße, verfügbaren Laborplätzen, Anwesenheitspflicht im Labor, typischen Gestaltung des Stundenplans, Verfügbarkeit von Labormitarbeitern, Laborinfrastruktur, Credit Points (5 CP) und ähnliche Rahmenbedingungen haben wir mit Architekturvorgaben gleichgesetzt. Auch wenn uns diese Rahmenbedingungen geläufig waren, haben wir sie zur Reflexion in unserem Dokument aufgeführt. Weiterhin liefern sie für Außenstehende (z. B. andere Departments/Studiengänge) Hinweise darauf, inwieweit unsere Umsetzungsideen übernommen werden können. Für den eigenen Überblick wurden auch Referenzen auf bestehendes Material aus anderen Modulen aufgeführt, sowie die Liste der Deliverables, d. h. Artefakte an Vorlesungsunterlagen, Übungsaufgaben, Laboraufgaben, Templates, usw. bis hin zur Klausur, erstellt. Eine Liste an Quellen wie Bücher oder Vod-/Podcasts ergänzt die Materialliste.

Requirements Engineering

Ziel eines Moduls ist, dass die Studierenden am Ende über einen Satz an (akademischen) Kompetenzen verfügen. Entsprechend wurden im Sinne von Requirements zu erreichende Kompetenzen formuliert. Es wurde hier das akademisches Kompetenzmodell nach Reis (2013) verwendet, welches keine weitere Unterteilung in fachliche, Methoden-, soziale und Selbstkompetenz (Böttcher u. a. (2011)) wie aus der Berufsbildungsforschung vorsieht. Die Kompetenzen wurden als Learning Outcomes (Biggs u. Tang (2007) oder Kennedy (2007)) formuliert, die ein Handlungsziel in einer komplexen Situation sichtbar machen. Im Sinne des Constructive Alignment (vgl. z. B. Biggs u. Tang (2007) oder Brabrand (2008)) lassen sich so leicht formative als auch summative Prüfungsfragen (Zürich (2008)) ableiten. Das Constructive Alignment sieht vor, dass die Intended Learning Outcomes mit den Prüfungen und die darauf vorbereitenden Lehreinheiten zueinander inhaltlich und auf den Niveaustufen (Bloom (1972)) kohärent sind. Es darf nicht erwartet werden, dass die Studierenden UML Modelle erstellen können, wenn in der Vorlesung nur die einzelnen Symbole durchgearbeitet werden und dann auf dieser Basis in der Prüfung Analyseaufgaben auf einem gegebenen Diagramm gestellt werden. Dieses wäre im Sinne des Constuctive Alignment inkohärent, alleine schon weil die drei Teile auf verschiedenen Niveaustufen liegen. Wird eine Modellierungskompetenz erwartet, dann ist eine Prüfungsaufgabe zur Modellierung zu stellen und diese Kompetenz (nicht die Musterlösung) im Rahmen der Lehreinheiten einzuüben. Das Design der Vorlesung arbeitet somit auf die Prüfung der Kompetenz hin.

Im Gegensatz zur Forderung von Oliver Reis wurden hier nicht nur Learning Outcomes im engeren akademischen Sinne formuliert, sondern auch Kompetenzen auf niedrigen Niveaustufen explizit aufgeführt. Es wurde so versucht, die zu erreichende Niveaustufe (Bloom (1972)) direkt in den verwendeten Verben abzubilden. Diese führten dann zu Unterkompetenzen analog zu Hierarchien von Requirements. Die Studierenden müssen zum Beispiel nicht in der Lage sein ein Lasten- oder Pflichtenheft zu erstellen; sie sollten aber die Bedeutung dieser Dokumente kennen. Dieses ”kennen” wurde durch die Unterkompetenzformulierung „Die Studierenden können den Unterschied zwischen einem Lasten- und einem Pflichtenheft erläutern.“[1] abgebildet. Im Gegensatz dazu wurde auf das Dokumentieren einzelner Requirements Wert gelegt, was sich im Rahmen des Themas Requirement Engineering in der Formulierung der Teilkompetenz „Die Studierenden können einzelne Requirements strukturiert dokumentieren und die Qualität der Dokumentation bewerten.“ widerspiegelt. Gerade die Bewertung der Qualität liegt auf einer hohen Niveaustufe. Dieses Vorgehen bei der Erstellung der Kompetenzliste führte zu Unterkompetenzen analog zu Hierarchien bei Requirements. Die Unterkompetenzen beschreiben hier detaillierter, was unter der übergeordneten Kompetenz verstanden wird oder führen auf, welcher Baustein zum Erlangen der Hauptkompetenz beitragen soll.

Beispiel aus dem Bereich Requirements Engineering:

Die Studierenden ...

K-10 ... können Kundenwünsche identifizieren und als Requirements-Katalog dokumentieren und bewerten.

K-10.1 ... können einzelne Requirements im Sinne von strukturiertem Text formulieren und dokumentieren.

K-10.2 ... können Requirements geeignet attribuieren (Prioritäten, Stakeholder, usw.).

K-10.3 ... können Abnahmekriterien (Abnahmetests) für Requirements entwerfen.

...

K-10.7 ... können die Qualität von Requirements beurteilen.

K-10.8 ... können strukturiert ein Requirements Review durchführen.

Für die Identifikation der erforderlichen Kompetenzen wurde wie folgt vorgegangen: Aus bestehenden Modulen der Informatik und der persönlichen Erfahrung des Software und Systems Engineering wurden die Hauptkompetenzen formuliert. Unterkompetenzen ergänzten diese oder beschrieben genauer, was die Hauptkompetenz eigentlich aussagt. Hierzu wurden auch Modulbeschreibungen anderer Hochschulen herangezogen. Weiterhin wurden Lehrbücher und Vorlesungsunterlagen gesichtet. Bei den interessanten Themenfeldern wurde immer die Frage gestellt, was für sichtbare Handlungen können die Studierende in diesem Themenfeld ausführen, bzw. was ist ein mög-

[1] Die abgeleitete Prüfungsfrage im Sinne des Constructive Alignment ist dann „Erläutern Sie die Unterschiede zwischen einem Lasten- und einem Pflichtenheft!“

liche Handlung, die sich aus dem Wissen erschließt? Die Antworten führten dann zur Formulierung der Kompetenz mit einem geeigneten Verb des sichtbaren Handelns[2].

Nichtfunktionale Anforderungen ergeben sich aus den planerischen Randbedingungen; dies bezieht sich zum einen auf den generellen Zuschnitt als Modul mit 3 Stunden Vorlesung und 1 Stunde Praktikum und die praktische Umsetzung wie oben beschrieben. Andererseits sind aber auch die Vorgaben der Prüfungsordnung bezüglich der Prüfungsform (Klausur) und die Prüfungsvoraussetzung durch das erfolgreiche Absolvieren aller Praktikumstermine und -aufgaben als einschränkende Randbedingungen für die didaktische Ausgestaltung des Moduls zu sehen.

Übergang zum Design

Beim Erarbeiten der Kompetenzen fielen automatisch bereits Zusatzinformationen an, die den Übergang zum Design, dem Ausgestalten von Vorlesung, Laborpraktika und Übungen, vereinfachen. Analog ergeben sich beim Erfassen von Requirements bereits Ideen zu möglichen Realisierungen oder Designs. Diese (Design-)Informationen wurden strukturiert in einem Mindmapping-Tool (hier FreeMind) erfasst. Zu jeder Kompetenz wurde die Motivation für die Aufnahme der Kompetenz, Verweise auf vorhandenes Material, Querverweise, Notizen, Ideen für mögliche Prüfungsaufgaben sowie Ideen zur Vermittlung der Kompetenz annotiert (vgl. Abbildung 1). Bei den Vermittlungsideen wurde neben Vorlesungsvorträgen auch schon Verweise auf Methoden der aktivierenden Lehre (Macke u. a. (2012) bzw. Dallmeier u. a. (2013)) im Rahmen des seminaristischen Unterrichts mit Aufgabenstellungen vermerkt. Es wurde trotz der scheinbar gleichmäßigen Struktur der Annotationen ein Mindmapping-Tool verwendet, da es wesentlich flexibler bei der Ausbildung von hierarchischen Strukturen ist, bzw. für die Aufnahme von Ideen besser geeignet ist als die strenge Tabellenstruktur eines Requirement Tools.

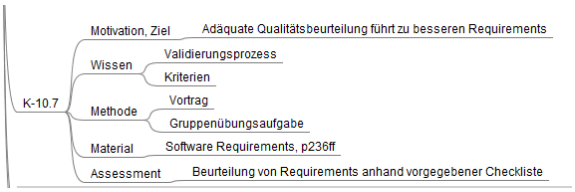


Abbildung 1: Zusatzinformationen zur Kompetenz 10.7 "... können die Qualität von Requirements beurteilen." im Mindmapping-Tool.

Im Sinne eines Architektur-Entwurfes wurde das Modul in thematischen Blöcke der beiden Lehrformen Vorlesung und Laborpraktikum aufgeteilt und grob der zeitliche Umfang pro Block festgelegt (Analogie: Assignment to Components). So wurden zum Beispiel für den Block „Requirements Engineering“ zwei Vorlesungstermine vorgesehen (siehe Tabelle 1).

In einer Matrix wurden dann die Kompetenzen den Blöcken zugeordnet. Es zeigt sich auch hier, dass einige Kompetenzen mehreren Vorlesungs- und Laborterminen zugeordnet werden können und müssen. Analog zum Verhalten, dass einige Requirements wie zum Beispiel Performance nur querschnittartig im System umgesetzt werden können, findet sich die Vermittlung einiger Kompetenzen als Querschnittsthemen wieder.

Die Zuordnung zu Vorlesungs- oder Laborterminen erfolgt je nach Kompetenz und erforderlichen Vermittlungs- als auch Prüfungsmethoden. Für die Kompetenz K-10.8 "Die Studierenden können strukturiert ein Requirements Review durchführen." ergibt sich durch das Verschieben des Verbs der Prüfungsauftrag "Führen Sie strukturiert ein Requirements Review durch." Durch die in der Prüfungsordnung vorgesehene Klausur (vgl. oben zu den nicht-funktionalen Anforderungen) lässt sich diese Kompetenz nur schlecht prüfen. Darüber hinaus sollten die zugehörigen Handlungen auch tatsächlich ausgeführt werden.

Wir hatten deshalb beschlossen, diese Kompetenz nicht im Sinne einer Notengebung zu prüfen. Diese Kompetenz wurde einem Labortermin zugeordnet, da hier Teams eine Aufgabe ausführen müssen und man als Dozent die Durchführung beobachten kann. Man kann die Teams bei der Ausführung beobachten und ihnen besser Feedback geben, als bei einer integrierten Übung in der Vorlesung. Weiterhin ist die Laborübung Teil der Prüfungsvorleistung und unterliegt somit auch einem Qualitätsanspruch. In den vorgelagerten Vorlesungen wird das Wissen um die Prozesse und die Qualitätsmerkmale von Reviews vermittelt. In dem eigentlichen Review im Praktikum sollen die Teams von vier Studierenden die Requirements eines anderen Teams des gleichen Labortermins überprüfen. Dies wiederum setzt voraus, dass vorher die Kompetenz zur Erstellung von Requirements vermittelt wurde. Aus dieser Abhängigkeit ergeben sich Randbedingungen für das Scheduling der Lehreinheiten und der Praktikumsinhalte.

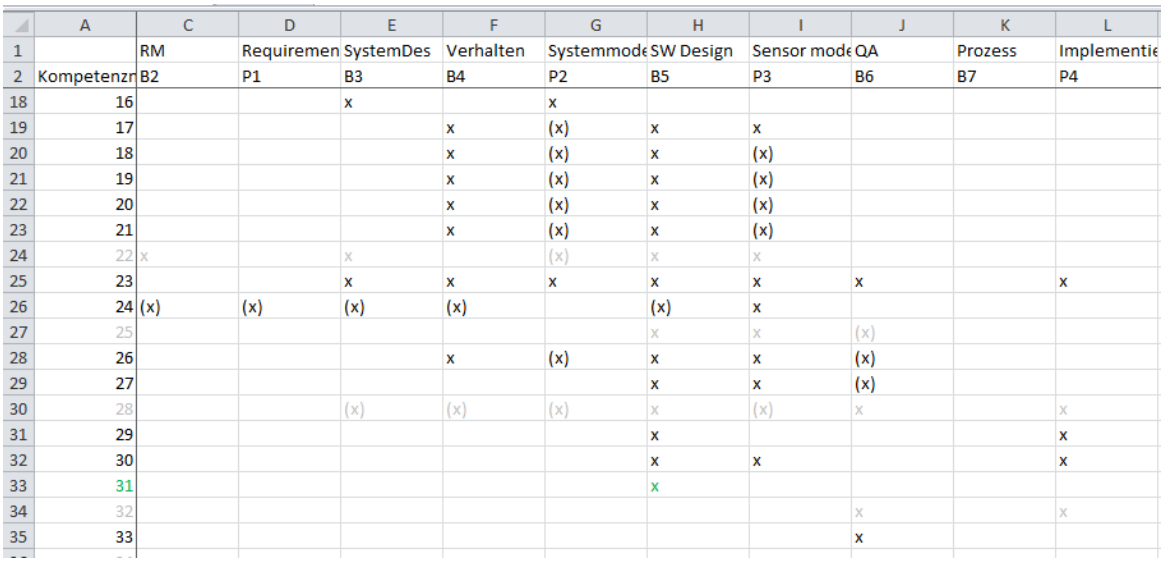
Insgesamt ergab sich für diesen Labortermin die Aufgabenstellung in das Erstellen von Requirements auf Basis einer Fallbeschreibung, welche eine kurzen Eingangsprüfung unterzogen wurden, und die Review-Sitzung unter Beobachtung mit abschließenden Feedbackrunden. Die Qualitätsbeurteilung und die überarbeiteten Requirements wurden dann geprüft und ergaben die Prüfungsvorleistung.

An diesem Punkt der Planung muss man abwägen, ob der Aufwand in Lehre und Prüfung zum Erreichen der jeweiligen Kompetenz gerechtfertigt ist. Falls nicht ist es erforderlich die geforderte Kompetenz anzupassen, da man sonst das Constructive Alignment nicht erzielt. Auch hier findet sich die Analogie zu den Aufgaben eines Softwarearchitekten wieder, der das Grob-

[2] Die Sichtbarkeit ist für die Ableitung einer guten Prüfungsaufgabe erforderlich.

Tabelle 1: Zuordnung Themenbereiche zu Vorlesungsterminen

Thema	Themenbereich	Termin(e)
Organisatorisches	Organisatorisches, Übersicht, Einführung	1
Requirements	Requirements, Kano-Modell, Stakeholder, Requirements Prozess, Dokumentation, Qualitätssicherung	1-2
Systemdesign	Systemdesign, Modellierung allgemein, SysML, Block Diagramms, Internal Block Diagrams, Flows, Systemsicht	3-4
Verhaltensmodelle	Verhaltensmodellierung, Sequence Diagrams, Activity Diagrams, State Diagrams	5-7
Software	Modellierung von Software, Datenmodelle, technische Systeme steuern, Umsetzung in lauffähige Software	7-9
QA	Qualitätssicherung	10-11
Prozesse	Prozesse/Configuration Management	11
Projektmanagement	Configmanagement, Ticketsysteme, Projektmanagement	12
Safety (Optional)	Prinziples, Begriffe, Analysemethoden, Techniken	12



	A	C	D	E	F	G	H	I	J	K	L
1		RM	Requiremen	SystemDes	Verhalten	Systemmode	SW Design	Sensor mode	QA	Prozess	Implementie
2	Kompetenzn	B2	P1	B3	B4	P2	B5	P3	B6	B7	P4
18	16			x		x					
19	17				x	(x)	x	x			
20	18				x	(x)	x	(x)			
21	19				x	(x)	x	(x)			
22	20				x	(x)	x	(x)			
23	21				x	(x)	x	(x)			
24	22	x		x		(x)	x	x			
25	23			x	x	x	x	x	x		x
26	24	(x)	(x)	(x)	(x)		(x)	x			
27	25						x	x	(x)		
28	26				x	(x)	x	x	(x)		
29	27						x	x	(x)		
30	28			(x)	(x)	(x)	x	(x)	x		x
31	29						x				x
32	30						x	x			x
33	31						x				
34	32								x		x
35	33								x		

Abbildung 2: Ausschnitt der Zuordnungsmatrix von Kompetenz zu Lehreinheiten.

design auf Machbarkeit im Rahmen der zur Verfügung stehenden Ressourcen prüfen und ggf. Maßnahmen ergreifen muss.

Implementierung

Auf Basis der Blockzuordnung der Kompetenzen und der Zusatzinformationen wurden die einzelnen Termine bezüglich ihrer Inhalte, Vermittlungsmethoden und zeitlichem Umfang geplant. Dabei wurde Material (Folien, Übungsaufgaben, usw.) aus bestehenden Vorlesungen übernommen, angepasst oder neu erstellt. Als ergänzendes Material wurden den Studierenden die Liste der zu erwerbenden Kompetenzen, Kontrollfragen und weiterführende Quellen für jeden Themenblock zur Verfügung gestellt. Ein Tracing der Kompetenzen auf einzelne Folien oder Übungen wurde als zu feingranular angesehen.

Bei der Umsetzung wurde bei den im Sinne des seminaristischen Unterrichts in die Vorlesungen integrierte Gruppenübungen darauf geachtet, dass in der Leistungsstärke gemischte, immer neu zusammengestellte Teams aktiviert wurden. Die Übungen sind großteils nach Ansätzen des Peer Facilitated Learning konzipiert, d. h. jeder Studierender skizziert zunächst eine individuelle Lösung, diese wird im zusammengestellten Team diskutiert und zu einem Teamergebnis zusammengeführt. Die Ergebnisse der Teams wurden dann jeweils auf einem ”Marktplatz” präsentiert und diskutiert. Gerade bei Modellierungsaufgaben erweist sich die Diskussion auf dem Marktplatz als günstige formative Lernkontrolle, da es oftmals keine eindeutige (Muster-)Lösung gibt.

Die Entwicklung der Laboraufgaben gestaltete sich schwieriger, da hier mehr Randbedingungen, insbesondere die verfügbaren oder beschaffbaren technische Aufbauten mit berücksichtigt werden müssen. Weiterhin darf die Problemstellung eine gewissen Komplexität weder über- noch unterschreiten, da die Problemstellungen alle Teammitglieder auslasten sollen.

Die Rahmenbedingungen des Departments sehen vier Laborterminen mit je 4 SWS vor. Dazu kommen Vor- und Nachbereitung. Typischerweise wird in unseren Modulen in Gruppen von 12-16 Studierenden in Zweierteams gearbeitet. Softwareengineering Methoden kommen aber erst dann zum Tragen, wenn eine Grundkomplexität überschritten wird. Somit wurde die Teamgröße im ersten Durchlauf auf 4-5 Studierende festgelegt und in nachfolgenden Durchläufen wieder auf 3-4 Studierende reduziert (vgl. Liebehenschel (2013)). In dieser Teamgröße können ausreichend komplexe Aufgaben gestellt werden, die in der Zeit bearbeitbar bleiben. Die Teamgröße erfordert eine andere Kommunikation als im eingespielten Zweierteam. Ein strukturiertes Vorgehen und ein Informationsaustausch über Diagramme (UML) wird nun erforderlich.

Die Labortermine eignen sich gut, die Studierenden beim Bearbeiten der Problemstellung und ihre Vorgehensweise zu beobachten. Deshalb wurden die Problemstellungen so ausgewählt, dass der Prozess im Vordergrund steht und das Ergebnis der Problembearbeitung durch nachzureichende Artefakte geprüft wurde. Im Einzelnen:

1. Labortermin: Strukturiertes Requirements Review

   Vorleistung: Kleines Requirements-Dokument erstellen

   Nacharbeiten: Einarbeiten der Mängelliste (erstellt durch anderes Team)

2. Labortermin: Modellierung eines Systems

   Vorleistung: Einarbeiten in die Diagrammtypen

   Nacharbeiten: Diagramme vollständig mit Kommentaren

3. Labortermin: Wartung Teil 1.

   Eine bestehende Steuerungssoftware soll erweitert werden. Aufarbeiten der Requirements und Analyse des Bestandssystems.

   Nacharbeiten: Dokumentation der Requirements und Analyseergebnisse

4. Labortermin: Wartung Teil 2.

   Design, Umsetzung und Test der Erweiterung.

   Nacharbeit: Dokumentation des Designs, der Umsetzung und der Abnahmetests

Gerade die Wartungsaufgabe erwies sich für die Studierenden als große Herausforderung, da sie sich in eine fremdes System eindenken müssen und sie mit dem üblichen Trial-and-Error nicht effektiv zum Ziel kommen. Trotz des Aufwandes und den Schwierigkeiten bei der Aufgabenstellung war das Feedback zu dieser Problemstellung sehr positiv, da es sich wie ein Problem aus dem realen Leben anfühlte.

Qualitätskontrolle

In der Qualitätskontrolle muss man zwischen der abschließenden Prüfung (weiter unten), Beurteilung des Lernfortschrittes im laufenden Semester und der Qualitätskontrolle in der Umsetzung unterscheiden.

Für die Beurteilung des Lernfortschrittes der Semestergruppe wurden die Diskussionen im seminaristischen Unterricht und die Bearbeitung der Übungsaufgaben herangezogen. Während der integrierten Gruppenübungen in den Vorlesungen hat man als Dozent die Möglichkeit zwischen den Teams zu wandern und die Arbeit der Teams zu beobachten. Hier kann man gut den Stand der Studierenden erkennen und auch direkt Feedback geben, falls die Studierenden kollektiv falsche Lösungsansätze verfolgen. Dies gilt ebenso bei den Diskussionen der Teamergebnisse. Formative Prüfungen durch Classroom Response Systeme, wie Klicker-Systeme, kamen hier noch nicht zum Einsatz, wären aber denkbar.

Im Bereich der Umsetzung wurde nach jedem Vorlesungstermin geprüft, ob die angesetzte Zeit für die Vermittlung der Teilkompetenz durch Vorlesungsteil und integrierten Übungen grob eingehalten wurde. Hier zeigte sich, dass der Zeitaufwand für die Gruppenübungen unterschätzt wurde, insbesondere die Zeit für die Diskussion der Ergebnisse. Letztere werden aber von uns als wesentlich für die Verankerung oder Vertiefung der Themen angesehen.

Fehler in den Unterlagen wurden direkt als Errata korrigiert. Weiterhin wurde ein Tagebuch geführt, in dem Beobachtungen aus den Veranstaltungen und Verbesserungsideen festgehalten wurden. Von Seiten der Hochschule werden regelmäßig Evaluationen der einzelnen Vorlesungen durchgeführt, so auch für dieses Modul. Hier waren insbesondere die Freitextrückmeldungen der Studierenden hilfreich, um Anpassungen vorzunehmen.

Maintenance

Die Vorlesung wurde nun bereits zwei Mal, unser Meinung nach erfolgreich, durchgeführt. Nach jedem Durchlauf wurden zunächst die Verbesserungsideen und die Beobachtungen aus dem Tagebuch ausgewertet und zusammengefasst. Analog zum prinzipiellen Vorgehen in der Softwareentwicklung müssen hier bei neuen Erkenntnissen aus dem Feld die Requirements überarbeitet werden, das Design angepasst werden, usw. Entsprechend führten die gewonnenen Erkenntnisse zur Überarbeitung der Kompetenzen, der Zusatzinformationen, der Laboraufgaben, bis hin zur Neugestaltung einzelner Folien oder Vorlesungsabschnitte. Die beobachteten Schwierigkeiten bei der Wartungsaufgabe führten zum Beispiel dazu, dass das Thema ”einarbeiten in ein fremdes System” nun explizit in der Vorlesung thematisiert wird.

Prüfung

Abgeschlossen wird das Modul mit einer schriftlichen Prüfung am Ende der Vorlesungen, wie es die Prüfungsordnung vorschreibt. Ein erfolgreicher Abschluss aller Labortermine ist Voraussetzung für die Zulassung zur Klausur. Eine individuelle Benotung ist im Laborpraktikum nicht erforderlich, nur eine individuelle Zulassung zur Klausur. Für das erfolgreiche Abschließen eines Labortermins wurden die abgelieferten Artefakte als Teamleistung bewertet. Auch wenn nur eine Zulassung erreicht werden muss, somit ein Mindestmaß an Leistung, so wurden auch hier beobachtet, dass es innerhalb einzelner Teams Ärger bezüglich der Zuarbeit einzelner Teammitglieder gab.

Die abschließende Klausur ist weiterhin klassisch mit kleinen Aufgaben aufgebaut und akkumuliert Teilpunkte, die wiederum auf eine Note abgebildet werden. Bei der Entwicklung der Klausuraufgaben konnten Aufgaben leicht aus den Kompetenzen abgeleitet werden. Aus der Kompetenz K-10.7 ”Die Studierenden können die Qualität von Requirements beurteilen.” wurde die Prüfungsaufgabe ”Beurteilen Sie die nachfolgenden Requirements auf ihre Qualität.” Dazu war ein Block mit Requirements gegeben, die in der Vorlesung angesprochene Mängel unterschiedlichster Art enthielten.

Bei Aufgaben höherer Niveaustufen, wie zum Beispiel die Qualitätsbewertung von gegebenen Requirements oder Modellierung, wurde in den Teilaufgaben ein Niveaustufenmodell angewandt. Es wurde nicht die Anzahl der gefundenen Problemstellen (ein Ding, ein Punkt), sondern die Komplexität und Tiefe der Analyse bewertet und auf eine Punktzahl abgebildet (Qualität der Antwort). Wurden zum Beispiel bei der Qualitätskontrolle der Requirements ausschließlich die Rechtschreibfehler bemängelt, aber nicht erkannt, dass immer wieder Synonyme verwendet wurden oder das es Widersprüche gab, so führte dieses zu einem schlechteren Ergebnis.

Ein vollständiger Übergang zur kompetenzorientierten Prüfung würde zwei bis drei komplexe Learning Outcomes/Kompetenzen prüfen, welche die notwendigen Teilkompetenzen mit abdecken. Ein derartiger Umstieg der Prüfungsmodalitäten wurde als zu großer Bruch betrachtet und wird nach und nach durch Abkehr von kleinen Teilaufgaben zu wenigen komplexeren Aufgaben vollzogen. Als Zwischenstufe wird eine Klassifikation der Teilaufgaben, wie von Waffenschmidt (Waffenschmidt (2013)) vorgeschlagen, mit entsprechendem Bewertungsschema angestrebt.

Fazit

Die hier gezeigte Übernahme von Vorgehensmodellen des Softwareengineerings erscheint vielleicht ungewöhnlich, hat uns aber einen guten Überblick verschafft und Hinweise für das jeweils weitere systematische Vorgehen gegeben.

Durch die gute Formulierung von (Teil-)Learning Outcomes war es zum Teil sehr leicht, Prüfungsaufgaben zu entwickeln. Auch konnte man den Studierenden Hinweise auf die Prüfung geben, insbesondere da die zur Verfügung stehenden Klausuren der Vergangenheit nicht mehr zur aktuellen Ausgestaltung des Moduls passten. Weiterhin hat die Kompetenzorientierung mit dem Hinterfragen der erwarteten Handlungsfähigkeit dazu geführt, dass sich weniger „sinnloses“ Wissen in der Vorlesung anhäuft als bei einer themenorientierten Vorgehensweise.

Der Ansatz zur Übernahme von Methoden aus dem Softwareengineering wurde noch nicht konsequent umgesetzt. So fehlt zum Beispiel eine Betrachtung des Kontextes der Vorlesung, d. h. die genaue Betrachtung aller anderen Vorlesungen des Curriculums. Auch wurde kein konsequentes Controlling bezüglich der eingeflossenen Aufwände durchgeführt[3]. Ein wesentlicher nächster Schritt wäre nun auch das sich Lösen von den gegebenen Rahmenbedingungen und zeitlichen Taktungen. Diese schränken das Denken in möglichen Vermittlungsmethoden noch zu sehr ein.

Die mit der Kompetenzorientierung einhergehende Problemorientierung war gerade in den Laboren eine Herausforderung für die schwächeren Studierenden. Sie erwarten klar umrissene (Teil-)Aufgaben, die abzuarbeiten sind. Die gefühlte Akzeptanz von realistischen Problemstellungen statt Aufgaben war bei den leistungsstärkeren, bzw. softwareaffineren Studierenden hoch und wurde begrüßt. Die Laboraufgaben waren so konzipiert, dass diese nur im 4er-Team bei strukturiertem Vorgehen in der Zeit zu lösen waren. Ein ”Gebastel”, wie man es oftmals im dritten Semester noch beobachtet, führte nicht zum Erfolg und ließ einige Teams dann doch auf systematisches Arbeiten umschwenken.

Die an das Softwareengineering angelegte systematisch konstruktive Vorgehensweise kann unserer Meinung nach auch für andere Module angewandt werden. Das Lecture Engineering gibt Dozenten aus den Ingenieurwissenschaften/der Informatik Mittel aus Ihrer Denkwelt an die Hand, um ihre Module durchzuplanen und auszugestalten. Die Qualität der Module erhöht sich, da eine Transparenz existiert und ein Abgleich im Sinne des Constructive Alignment zwischen Learning Outcome, Vorlesungs- und Übungsinhalten und der Prüfung vereinfacht wird.

Literatur

[Biggs u. Tang 2007] BIGGS, J. ; TANG, C.: *Teaching for Quality Learning*. McGraw-Hill Companies,Inc., 2007

[Bloom 1972] BLOOM, Benjamin S. ; BLOOM, Benjamin S. (Hrsg.): *Taxonomie von Lernzielen im kogni-*

[3] Falls Leser sich auf einen ähnlichen Weg begeben, wären wir an entsprechenden Daten interessiert.

tiven Bereich. 4. Beltz Verlag, Weinheim und Basel, 1972

[Böttcher u. a. 2011] BÖTTCHER, Axel ; THURNER, Veronika ; MÜLLER, Gerhard: Kompetenzorientierte Lehre im Software Engineering. In: LUDEWIG, Jochen (Hrsg.) ; BÖTTCHER, Axel (Hrsg.): *Software Engineering im Unterricht der Hochschulen (SEUH 2011)*. München : dpunkt.verlag, Heidelberg, 2011, S. 33–39

[Brabrand 2008] BRABRAND, Claus: *Teaching Teaching & Understanding Understanding*. DVD, 2008

[Dallmeier u. a. 2013] DALLMEIER, Valentin ; GROSS, Florian ; HAMMACHER, Clemens ; HÖSCHELE, Matthias ; JAMROZIK, Konrad ; STREIT, Kevin ; ZELLER, Andreas: Muster der Softwaretechnik-Lehre. In: SPILLNER, Andreas (Hrsg.) ; LICHTER, Horst (Hrsg.): *Software Engineering im Unterricht der Hochschulen (SEUH 2013)*. Aachen : dpunkt.verlag, Heidelberg, 2013, S. 101–102

[Hammerschall u. Beneken 2013] HAMMERSCHALL, U. ; BENEKEN, G.H.: *Software Requirements*. Pearson Studium, 2013. – ISBN 9783868941517

[Hummel 2013] HUMMEL, Oliver: Transparente Bewertung von Softwaretechnik-Projekten in der Hochschullehre. In: SPILLNER, Andreas (Hrsg.) ; LICHTER, Horst (Hrsg.): *Software Engineering im Unterricht der Hochschulen (SEUH 2013)*. Aachen : dpunkt.verlag, Heidelberg, 2013, S. 103–114

[Kennedy 2007] KENNEDY, D.: *Writing and Using Learning Outcomes: A Practical Guide*. University College Cork, 2007. – ISBN 9780955222962

[Liebehenschel 2013] LIEBEHENSCHEL, Jens: Software-Engineering Projekte in der Ausbildung an Hochschulen - Konzept, Erfahrungen und Ideen. In: *Software Engineering im Unterricht der Hochschulen (SEUH 2013)*. Heidelberg : .verlag, 2013, S. 27–34

[Lotter 2005] LOTTER, Wolf: DER ROTE FADEN. In: *Brandeins* 02 (2005)

[Macke u. a. 2012] MACKE, G. ; HANKE, U. ; VIEHMANN, P.: *Hochschuldidaktik: Lehren – vortragen – prüfen – beraten.* Beltz, 2012

[Reis 2013] REIS, Oliver: Kompetenzorientierte Prüfungen: Prüfungstheorie und Prüfungspraxis. In: GUTGE-WICKERT, Angelika (Hrsg.) ; KESSLER, Ulrike (Hrsg.): *Die homöopathische Behandlung chronischer Krankheiten*, 2013

[Waffenschmidt 2013] WAFFENSCHMIDT, Eberhard: Kompetenzorientierte schriftliche Prüfungen. In: *Handbuch Qualität in Studium und Lehre* (2013)

[Zürich 2008] ZÜRICH ; HOCHSCHULDIDAKTIK, Universität Zürich. A. (Hrsg.): *Leistungsnachweise in modularisierten Studiengängen: Dossier*. Universität Zürich, Arbeitsstelle für Hochschuldidaktik, 2008

Koordination textbasierter synchroner Kommunikation als Kompetenz im Software Engineering

Kerstin Raudonat, Hochschule Heilbronn

kerstin.raudonat@hs-heilbronn.de

Dominikus Herzberg, Technische Hochschule Mittelhessen

dominikus.herzberg@mni.thm.de

Nicola Marsden, Hochschule Heilbronn

nicola.marsden@hs-heilbronn.de

Zusammenfassung

Kommunikation in verteilten Teams im Software Engineering ist an die Möglichkeiten medial vermittelter Kommunikation gebunden. Für eine erfolgreiche Zusammenarbeit sind Faktoren wie die Koordination der Kommunikation von Bedeutung. Kompetenzen zur Kommunikation und Koordination gilt es auch in der Software-Engineering-Ausbildung zu verankern. Dazu ist es notwendig, entsprechende Kommunikationspraktiken zu verstehen und für die Lehre im Software Engineering zugänglich zu machen. Hierzu wurde eine Untersuchung zu kommunikativen Koordinationsleistungen durchgeführt, die Hinweise zu kommunikativen Praktiken gibt. Unterschieden wird nach Erfahrungen mit medialer Kommunikation in Online-Rollenspielen, da Online-Rollenspielende zur erfolgreichen Bewältigung von Spielinhalten in Gruppen ebenfalls koordinative Leistungen in vermittelter Kommunikation erbringen müssen. So wird der Frage nachgegangen, wie Gruppen kommunizieren und welche Aspekte davon erfolgreich und für den Bereich der Softwareentwicklung adaptierbar sind.

Einführung

Software Engineering findet aufgrund der Komplexität des nötigen Wissens und des Umfangs der zu erstellenden Systeme fast ausschließlich in Teams bzw. in anderen Zusammenhängen statt, in denen Kommunikation und Koordination mit anderen Personen für die erfolgreiche Erstellung des Softwaresystems wichtig sind. Neben dem Beherrschen der Technologien und Werkzeuge für Software Engineering spielen auch nicht-fachliche Kompetenzen wie Kommunikation und die Koordination der Beteiligten untereinander eine Schlüsselrolle. Software-Engineering-Ausbildung sollte deshalb Fähigkeiten nicht nur auf theoretischer und technischer Seite, sondern auch in nicht-fachlichen Kompetenzbereichen aufbauen (Böttcher, Thurner & Müller, 2011).

Um diesen Aufbau von Kompetenzen zur Kommunikation und Koordination im Software Engineering praxisnah und berufsbefähigend zu vermitteln, müssen zwei zentrale Entwicklungen im Bereich der Softwareentwicklung berücksichtigt werden: Zum einen wird Software zunehmend in weltweit verteilten Teams entwickelt, zum anderen finden agile Methoden der Softwareentwicklung zunehmend Verbreitung (Boden, 2012). Dabei sind diese Entwicklungen in der Kombination durchaus problematisch: Agile Verfahren legen großen Wert auf das Setzen von zeitlichen Rahmenbedingungen („time-boxing“), z.B. in Form von Sprints im Scrum, aber auch in der zeitlichen Limitierung von einzelnen Meetings (Meyer, 2014), stellen also erhöhte Anforderungen an Kommunikation unter zeitkritischen Bedingungen. Folgerichtig betonen sie die Wichtigkeit der Face-to-Face-Kommunikation und stellen dar, dass verteilte Zusammenarbeit im Team möglichst vermieden werden sollte (Larman & Vodde, 2010). Allerdings führen die anhaltenden Trends hin zur Globalisierung und zur Agilisierung (Hanssen, Smite & Moe, 2011) dazu, dass agile Methoden zunehmend auch medial vermittelt zum Einsatz kommen (Eckstein, 2013; Hossain, Bannerman & Jeffery, 2011; Sutherland, Viktorov, Blount & Puntikov, 2007). Um den Entwicklungen hin zu agilen Methoden in der verteil-

ten Softwareentwicklung in die Ausbildung im Software Engineering zu integrieren, wurden bereits erste Ansätze für die Lehre entwickelt (z.B. (Paasivaara, Lassenius, Damian, Raty & Schroter, 2013), jedoch wurde die Vermittlung von Koordinationskompetenz in der Lehre des Software Engineering dabei bisher nicht näher in den Fokus genommen.

Beim Einsatz von agilen Methoden in der verteilten Entwicklung zeigt sich jedoch deutlich, dass speziell die Koordination innerhalb von und zwischen den Teams zu den zentralen Herausforderungen gehören (Alzoubi & Gill, 2014; Schneider, Torkar & Gorschek, 2013). Um dieser wichtigen Rolle von Koordination gerecht zu werden und sie auch als Kompetenz im Software Engineering vermitteln zu können, soll dieser überfachliche Kompetenzbereich näher untersucht und die unterschiedlichen Ausprägungen dieser Kompetenz bei den Studierenden strukturiert werden.

Um diesen für die verteilte agile Softwareentwicklung wichtigen Kompetenzbereich mit den dahinter stehenden Kommunikationspraktiken zu verstehen und somit für die Lehre im Software Engineering zugänglich zu machen, standen folgende Fragen im Vordergrund:

1. Welche Elemente der Koordinationskompetenzen lassen sich bei der verteilten kooperativen Aufgabenbearbeitung von Studierenden unterscheiden?
2. Welche Unterschiede machen Erfahrungen mit synchroner textbasierter Kommunikation im Kontext von Online-Rollenspielen hinsichtlich der Koordinationsleistung?

Hierzu wurde eine Untersuchung zu kommunikativen Koordinationsleistungen bei verteilter kooperativer Aufgabenbearbeitung durch Studierende durchgeführt. Dabei wurde zwischen Studierenden mit und ohne Erfahrung in Online-Rollenspielen unterschieden, da entsprechende Koordinationskompetenzen in synchroner textbasierter Zusammenarbeit auch dort zum Tragen kommen und entsprechend unterschiedlich ausgeprägt sein können.

Im Folgenden werden zunächst grundlegende Begriffe der Koordination und Gruppenkommunikation dargestellt und es werden Koordinationsanforderungen in Online-Rollenspielen beleuchtet. Dann werden die Methode und die Ergebnisse der Untersuchung berichtet, abschließend werden die Ergebnisse diskutiert und Implikationen für den Software-Engineering-Unterricht abgeleitet.

Koordination und Gruppenkommunikation

Wichtige Aspekte für gelingende Gruppenkommunikation, die unter medialen Bedingungen besondere Beachtung finden, sind u.a. die Herstellung und Aufrechterhaltung von Kohärenz (im Sinne einer inhaltlichen Verbundenheit von Aussagen), die Gewährleistung von gegenseitigem Verständnis (Grounding) und die Koordination der Kommunikation. Gerade in synchroner textbasierter Kommunikation wie z.B. im Chat können – durch unterschiedliche Interaktionsgeschwindigkeit und parallele Eingaben – Wartezeiten oder zerstückelte Dialoge entstehen, die der Kohärenz abträglich sein können. Koordination kann dies beispielsweise abmildern, indem das turn-taking strukturiert wird.

Koordination ist ein wichtiger Aspekt von Zusammenarbeit und Kommunikation unter medialen Bedingungen (Paechter, 2001), insbesondere auch in der agilen Softwareentwicklung (Pries-Heje & Pries-Heje, 2011; Strode, Huff, Hope & Link, 2012). Der Begriff der Koordination kann sich im Bereich der Kommunikation auf zwei Ebenen beziehen. Einerseits bedarf der Kommunikationsprozess selbst der Koordination (z.B. Steuerung der Sprecherwechsel), andererseits kann Koordination auch auf inhaltlicher Ebene stattfinden. Diese Art der Koordination kann als „the act of managing interdependencies between activities performed to achieve a goal“ (Malone & Crowston 1990, S. 361) beschrieben werden. Sie ist demnach auf die konkreten Aufgaben und Ziele bezogen, die der Kommunikationssituation zugrunde liegen. Im Folgenden werden kommunikative Handlungen, die der Koordination dienen, als *kommunikative Koordinationsleistungen* bezeichnet. Eine funktionierende Kommunikation ist dabei Voraussetzung, um eine effektive Koordination aushandeln zu können.

Damit Kommunikation in verteilten Teams zu qualitativ hochwertigen Arbeitsergebnissen führen kann, muss zunächst das gegenseitige Verständnis der Gruppenmitglieder gewährleistet sein. Zudem hängt die Gruppenleistung davon ab, wie die Mitglieder die gemeinsame Arbeit im Diskurs organisieren. Basierend auf Paechters (2003) Modell der aufgabenbezogenen Kommunikation lassen sich dabei verschiedene Aspekte des medial vermittelten Diskurses in Gruppen unterscheiden (vgl. Abb. 1): Dabei dient das *Grounding*, also die Verankerung der Kommunikation in gemeinsamem Wissen, gemeinsamen Vorannahmen und gemeinsamen Werten (Clark & Brennan, 1991; Walther, 2012), als Grundlage und bietet die grundsätzliche Möglichkeit zur Kommunikation. Darauf aufbauend gibt es Informationsmitteilung, Informationselaboration und Koordination als Kommunika-

tionspraktiken zur verbalen Organisation der gemeinsamen Arbeitsprozesse. Schließlich wird der Austausch und die Koordination zur Technik unterschieden, wobei mit Technik an dieser Stelle die für die medial vermittelte Kommunikation nötige Technik, also beispielsweise der Chat-Client, gemeint ist.

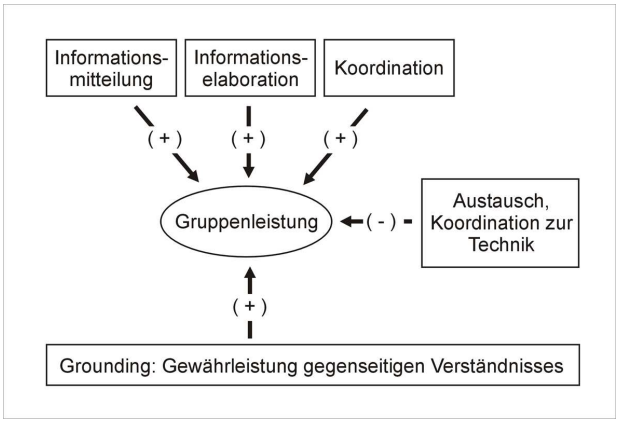


Abb. 1: Zusammenhang zwischen Grounding-Prozessen, aufgabenorientierter und technischer Kommunikation und der Gruppenleistung (Paechter, 2003, S.112)

Die empirische Überprüfung des Modells zeigte, dass in synchroner textbasierter Kommunikation insbesondere die Kohärenz zur Qualität der Arbeitsergebnisse beitrug. Dagegen zeigten sich beispielsweise in Videokonferenz und Präsenztreffen Diskursinhalte wie Elaboration oder Koordination als besonders bedeutsam (Paechter, 2003).

Koordinationsanforderungen in Online-Rollenspielen

Im Bereich der Online-Rollenspiele sind Anforderungen an Kommunikation und Koordination zum Teil dieselben, wie sie zuvor für die verteilten Teams erläutert wurden. Gerade in Massively Multiplayer Online Role-Playing Games (MMORPG) wie beispielsweise *World of Warcraft* oder *Final Fantasy XI* sind Zusammenarbeit und Kommunikation wichtige Elemente, da viele Spielinhalte nur in Gruppen zu bewältigen sind. Zur gemeinsamen und erfolgreichen Bewältigung von Spielaufgaben und -anforderungen müssen Spielende zum Teil hohe Koordinationsleistungen erbringen. In Relation zu einer steigenden Spieleranzahl in einer Gruppe steigen auch die Anforderungen an die Koordination.

Um den Umfang und die Komplexität der nötigen Koordinationsleistung eines Teams zu verdeutlichen, soll ein Beispiel für eine solche Spielsituation aus dem MMORPG *World of Warcraft* (*Mists of Pandaria*) erläutert werden: Die Instanz *Siege of Orgrimmar* ist ein zur Erscheinungszeit (2012) auf mindestens zehn Spielcharaktere der Maximalstufe ausgelegter Spielbereich, der besonders schwierige Gegner enthält – u.a. den Bossgegner *General Nazgrim*. Diesen zu besiegen, ist ohne abgesprochene Taktik nicht möglich. *General Nazgrim* kämpft in verschiedenen Haltungen, durch die er unterschiedliche Fähigkeiten nutzen kann, um den Spielcharakteren Schaden zuzufügen. Je nach Haltung darf er angegriffen werden oder nicht; daran müssen sich alle Spielenden halten, um der Gruppe nicht zu schaden. Alle 60 Sekunden wechselt er die Haltung und alle 15 Sekunden nutzt er eine besonders gefährliche Spezialfähigkeit, mit der die Spielenden auf eine verabredete Weise umgehen müssen. Zusätzlich erscheinen an vier Positionen im Spielraum alle 45 Sekunden weitere Gegner, die nach Absprache von festgelegten Spielenden dann möglichst schnell eliminiert werden müssen. Je nach Art der erscheinenden Gegner können unterschiedliche Spielcharaktere dafür eingeteilt werden, die sie nach einer abgesprochenen Prioritätenliste ausschalten. Auch müssen an verschiedenen Zeitpunkten einzelne Spielende den Platz bzw. die Aufgabe tauschen. Hinzu kommen weitere Details, die an dieser Stelle nicht weiter ausgeführt werden können. Es wird deutlich, dass alle Beteiligten wissen müssen, welche Aktionen zu welchem Zeitpunkt notwendig sind. So ist es erforderlich, dass die Spielenden vorab eine Taktik vereinbaren sowie Aufgaben verteilen und sich während der Kampfhandlungen gegenseitig anweisen und absprechen.

Spielende müssen diese koordinativen Anforderungen kommunikativ mittels der zur Verfügung stehenden, medialen Kommunikationsmöglichkeiten bewältigen. Es ist anzunehmen, dass Spielende, die häufig in Gruppen komplexe Spielaufgaben bewältigen bzw. dies versuchen, kommunikative Praktiken zur Koordination im Rahmen medialer Kommunikation etablieren. Studierende mit Erfahrungen in Online-Rollenspielen sind also geübt darin, eine Gruppenleistung in medial vermittelter Kommunikation zu erbringen. Entsprechend sollen sie betrachtet werden, um Anhaltspunkte für positive, hilfreiche Kommunikationspraktiken im Hinblick auf Koordination von synchroner textbasierter Gruppenkommunikation zu identifizieren und diese für die verteilte Softwareentwicklung zu adaptieren. Demzufolge wurden in der Untersuchung Studierende mit und ohne Vorerfahrungen in Online-Rollenspiele bei verteilter kooperativer Aufgabenbearbeitung im Hinblick auf Koordination analysiert.

Untersuchung zu kommunikativen Koordinationsleistungen bei kooperativer Aufgabenbearbeitung im Chat

Anhand der im Folgenden vorgestellten Untersuchung soll der Frage nachgegangen werden, inwieweit sich Spielergruppen und Nichtspielergruppen bei der gemeinsamen Bearbeitung einer Aufgabe im Chat im Hinblick auf kommunikative Koordinationsleitungen unterscheiden und inwiefern sie unterschiedlich erfolgreiche kommunikative Handlungsweisen wählen.

Gruppen und Aufgabenstellung

Es wurden vier Untersuchungsgruppen anhand der Bearbeitung einer kooperativen Aufgabe im Chat als medialer Kommunikationsumgebung verglichen. Zwei Gruppen entsprachen der Untersuchungsbedingung ‚Spieler' und bestanden aus jeweils drei Teilnehmenden, die kurz vor Ende ihres Studiums standen bzw. dieses vor kurzem abgeschlossen hatten und regelmäßig MMORPGs nutzen. Dies wurde mit einem Fragebogen überprüft, der ergab, dass alle Teilnehmenden der Bedingung ‚Spieler' mehrmals pro Woche und im persönlichen Schnitt mehr als zwanzig Stunden pro Woche MMORPGs spielen. Als am häufigsten genutztes Spiel wurde mehrheitlich ‚World of Warcraft' genannt. Diese beiden Gruppen werden im Folgenden als Spielergruppen bezeichnet. Zwei weitere Gruppen entsprachen der Untersuchungsbedingung ‚Nichtspieler' und bestanden ebenfalls aus jeweils drei Teilnehmenden, die kurz vor Ende ihres Studiums standen bzw. dieses vor kurzem abgeschlossen hatten, die aber angaben, niemals Online-Rollenspiele zu spielen oder gespielt zu haben.

Als Grundlage für die kooperative Arbeit diente den Gruppen eine vorgegebene Aufgabe, die es innerhalb einer Zeitbegrenzung zu bearbeiten galt. Jede Chatgruppe erhielt schriftlich dieselbe Arbeitsanweisung, die eine kurze Einweisung zum allgemeinen Ablauf (z.B. Bearbeitungszeit, Verweis auf Protokollierung des Chats), die zu bearbeitende Aufgabe sowie den Hinweis umfasste, die Aufgabe gemeinsam zu lösen und diese Lösung schriftlich in einem externen Dokument abzugeben. Zum einen ermöglichte die externe Lösungsabgabe einen Vergleich der im Chat erarbeiteten Lösung mit der schriftlich festgehaltenen. Zum anderen verdeutlichte dies den Teilnehmenden, dass sie sich auf ein Ergebnis einigen müssen. So sollte verhindert werden, dass das Gespräch im Chat sich zu einer ziel- und ergebnislosen Diskussion entwickelt, die mit dem Ende der Bearbeitungszeit einfach abgebrochen wird. Die zu bearbeitende Aufgabe war eine Postkorbaufgabe, die darin bestand, innerhalb einer bestimmten Zeitspanne eine vorgegebene Anzahl von Stationen (z.B. Supermarkt, Bahnhof) aufzusuchen. Dabei waren festgelegte Wegzeiten zwischen verschiedenen Stationen sowie Aufenthaltszeiten und Terminvorgaben (z.B. Abfahrtszeit eines Zuges) zu berücksichtigen. Es ist anzumerken, dass es nicht nur eine mögliche Lösung gibt. Für das Aufsuchen der einzelnen Stationen entsprechend der Vorgaben wurden Punkte verteilt, die für den Vergleich der Lösungen der einzelnen Gruppen notwendig sind. Die Bearbeitungszeit für die Aufgabe betrug 45 Minuten.

Untersuchungskategorien

Der Vergleich der Gruppen erfolgte anhand der Chat-Protokolle, die auf der Basis der Untersuchungskategorien analysiert wurden. Hierfür wurden im Vorfeld Kategorien zur Erfassung bzw. Beschreibung kommunikativer Koordinationsleistungen erarbeitet (vgl. Abb. 2). Zusätzlich zur Zuordnung der Kategorie wurde auch die Art der Äußerung erhoben, also ob sie verbaler Art oder paraverbaler bzw. nonverbaler Art war. Unter para- und nonverbalen Äußerungen sind nichtsprachliche Zeichen, Symbole und verschriftlichte nonverbale Signale zusammengefasst, die im Folgenden als *paralinguistische Ausdrucksformen* bezeichnet werden.

Abb. 2: Kategorien zur Erfassung kommunikativer Koordinationsleistungen

Kommunikationssteuerung: Um das Ausmaß der Kommunikationssteuerung – also die Anstrengungen zur Steuerung und Aufrechterhaltung der Kommunikation – zu erfassen, dienten die Kategorien ‚Ansprache' und ‚Bestätigung'. Zu ‚Ansprache' sind Äußerungen zu zählen, die einen Sprecherwechsel vorbereiten, indem beispielsweise eine Person namentlich angesprochen wird. Die Kategorie ‚Bestätigung' umfasst Aussagen, die – im Unter-

schied zu einer inhaltlichen Zustimmung – lediglich eine einfache Bestätigung darstellen. Die Sprecher machen auf diese Weise deutlich, dass sie eine Aussage eines anderen Sprechers zur Kenntnis genommen haben, ohne diese zu bewerten oder ihr inhaltlich zuzustimmen. In der Face-to-Face-Kommunikation kann diese Art der Bestätigung beispielsweise einem Nicken oder einem gemurmelten „hmm" entsprechen, das dem Gegenüber Aufmerksamkeit signalisiert.

Aufgabenorientierte Koordination: Zur Erfassung aufgabenorientierter Koordination dienten die Kategorien ‚Zielorientierung' und ‚Erfolgskontrolle'. Zu ‚Zielorientierung' zählen Äußerungen, die direkt auf die Aufgabenstellung und deren Vorgaben zurückverweisen und so eine zielorientierte Gesprächsführung unterstützen. Die Kategorie ‚Erfolgskontrolle' umfasst dagegen Aussagen, die eine Überprüfung von Vorschlägen und Zwischenergebnissen betrifft. Zur Feststellung der Aufgabenorientierung in einer Gruppe wird zudem grundsätzlich eine Kategorie ‚Inhalt' benötigt. Diese dient allerdings nicht der Erfassung von kommunikativen Koordinationsleistungen und ist daher in Abb. 2 nicht dargestellt. Sie trägt im Sinne einer Zusatzinformation der Verdeutlichung bei, wie viele der Beiträge der Darstellung von Inhalten dienen, ohne jedoch eine koordinierende Funktion zu besitzen.

Koordination der kooperativen Arbeit: Neben Kommunikationssteuerung und Aufgabenorientierung ist für die Arbeit in Gruppen von Bedeutung, inwieweit echte Kooperation stattfindet – also inwieweit alle Gruppenmitglieder in den Arbeitsprozess einbezogen werden, der Ablauf organisiert wird und das Ergebnis ein gemeinsames ist. Um die Anstrengungen zur Verwirklichung von Kooperation und Gemeinschaftlichkeit erfassen zu können, dienten die Kategorien ‚Bearbeitungsablauf', ‚Beitragssicherung', ‚Beteiligungssicherstellung' und ‚Zustimmung'. Zu ‚Bearbeitungsablauf' sind Aussagen zum allgemeinen Ablauf der Aufgabenbearbeitung zu zählen wie beispielsweise die Festlegung eines Schriftführers, der das Ergebnis festhält. Unter ‚Beitragssicherung' sind Zusammenfassungen bisheriger Ergebnisse einzuordnen. Die Kategorie ‚Beteiligungssicherstellung' umfasst Aussagen, die dazu dienen, andere Gruppenmitglieder einzubeziehen bzw. sicherzustellen, dass alle Mitglieder sich beteiligen und dem Verlauf folgen können. ‚Zustimmung' beinhaltet Aussagen, die eine Zustimmung auf inhaltlicher Ebene ausdrücken wie beispielsweise die Annahme von Vorschlägen.

Ergebnisse

Im Folgenden werden die Ergebnisse der Untersuchung vorgestellt und anschließend im Hinblick auf Implikationen für die Vermittlung von Koordinationskompetenzen bei der verteilten kooperativen Aufgabenbearbeitung im Software Engineering diskutiert. Vorab ist anzumerken, dass bei der Analyse zwischen den Begriffen ‚Meldung' und ‚Aussage' unterschieden wurde. Meldungen sind separat gesendete Beiträge der Teilnehmenden, die im Chat-Fenster als Einheiten erscheinen. Eine Meldung kann jedoch mehrere Aussagen beinhalten: Aussagen sind als Sinneinheiten zu verstehen, die einzelne Wörter, Wortgruppen oder Satzgefüge umfassen können. Diese Aussagen stellen als Kodiereinheiten die Grundlage der Kategorisierung dar. Die Summe der nach Kategorien kodierten Aussagen und der nicht zu kategorisierenden Aussagen stellt als Gesamtzahl aller erfassten Aussagen den Bezugswert für die Berechnung der relativen Häufigkeiten. So lässt sich eine Relation zwischen den jeweiligen Häufigkeiten in bestimmten Kategorien und der tatsächlichen Anzahl an getroffenen Aussagen im gesamten Gespräch der jeweiligen Gruppen herstellen. Die entsprechenden Rahmendaten sind in Abb. 3 dargestellt.

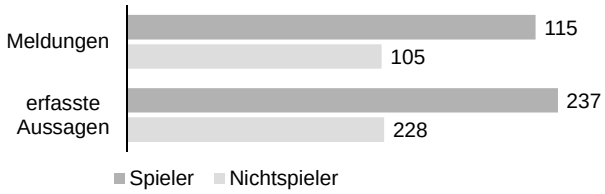


Abb. 3: Gesamtanzahl der Meldungen und der erfassten Aussagen

Bei den Spielern ergab sich eine durchschnittlich höhere Anzahl an erfassten Aussagen (176) als bei den Nichtspielern (105). Nach Auswertung der Lösungen verzeichneten die Spielergruppen auch einen im Schnitt höheren Punktwert (115 Punkte) als die Nichtspielergruppen (88 Punkte).

Die folgend dargestellte Auswertung beschränkt sich auf die Darstellung der relativen Häufigkeiten (prozentuale Anteile) und der Mittelwerte. Aufgrund des geringen Umfangs dieser exemplarischen Untersuchung (zwei Gruppen pro Untersuchungsbedingung) sind weitere statistische Berechnungen nicht zweckmäßig. Folglich kann hier die Signifikanz auftretender Unterschiede zwischen den einzelnen Gruppen nicht bewertet werden, aber es können interessante Hinweise und Tendenzen aufgezeigt werden.

Sprecherwechsel und Kohärenz

Um Aussagen bezüglich der Kohärenz treffen zu können, wurde die Häufigkeit der kohärenten und nichtkohärenten Sprecherwechsel erfasst. In diesem speziellen Fall beziehen sich die relativen Häufigkeiten auf die Summe aller Sprecherwechsel. Als

kohärent wurde ein Sprecherwechsel betrachtet, wenn aufeinanderfolgende Aussagen inhaltlich zusammenhängend waren – unter Beachtung der Möglichkeit, dass aufgrund von unterschiedlichen Schreibgeschwindigkeiten kleine Verschiebungen bzw. Unterbrechungen auftreten können. Abb. 4 stellt die kohärenten und nichtkohärenten Anteile der Sprecherwechsel dar.

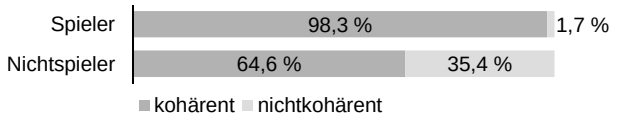


Abb. 4: Kohärente und nichtkohärente Sprecherwechsel

Bezüglich der Kohärenz beim Sprecherwechsel sind bei den untersuchten Gruppen deutliche Unterschiede festzustellen. Die Nichtspielergruppen haben deutlich mehr nichtkohärente Sprecherwechsel zu verzeichnen als die Spielergruppen (35,4% gegen 1,7%). Offensichtlich gab es in den Nichtspielergruppen deutlich mehr Probleme bei der Herstellung und Aufrechterhaltung der Kohärenz als in den Spielergruppen. Wie zuvor erläutert, hat die konversationale Kohärenz Einfluss auf die Grounding-Prozesse, die für eine effektive Zusammenarbeit in Gruppen von Bedeutung sind. Zudem hängen die Koordination der gemeinsamen Arbeit und die Lösungsfindung in einer Gruppe wesentlich von der Kohärenz des Diskurses ab. Es stellt sich die Frage, worin dieser deutliche Unterschied begründet ist.

Die qualitative Analyse der Beiträge zeigt, dass sich die Gespräche der Spielergruppen ohne Schwierigkeiten nachvollziehen lassen: Auf Aussagen folgen Reaktionen, Denkprozesse und Anmerkungen werden verbalisiert. Bei den Nichtspielergruppen dagegen werden häufiger Aussagen nicht beantwortet, vielleicht nicht einmal wahrgenommen, die Aussagen haben oftmals keinen direkten Bezug zueinander und enthalten Gedankensprünge. Hier fehlen offensichtlich Verbalisierungen und es ist zu vermuten, dass nicht jeder Aussage Aufmerksamkeit geschenkt wurde; dies mag auch an unterschiedlichen Interaktionsgeschwindigkeiten liegen. Es zeigt sich auf jeden Fall die Bedeutung der kontinuierlichen Verbalisierung und Überprüfung des aktuellen Standes im Chat, da Teilnehmende hier eben nicht einfach nebenher zuhören können wie es z.B. in einer Face-to-Face-Interaktion möglich ist.

Kommunikationssteuerung

Zur Erfassung der Anstrengungen zur Steuerung und Aufrechterhaltung der Kommunikation sind die Kategorien ‚Ansprache' und ‚Bestätigung' heranzuziehen. Abb. 5 stellt die relativen Häufigkeiten bezüglich dieser Kategorien dar.

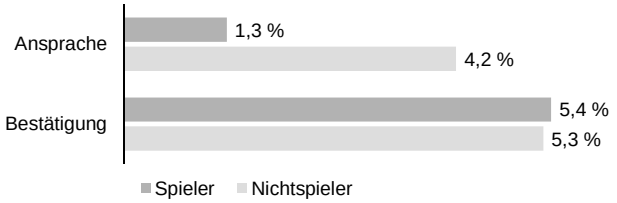


Abb. 5: Kommunikationssteuerung

In der Kategorie ‚Ansprache' weisen die Spielergruppen prozentual weniger Aussagen auf als die Nichtspieler (1,3% gegen 4,2%), in der Kategorie ‚Bestätigung' dagegen nur geringfügig. Bezüglich der Spielergruppen ist festzustellen, dass sie im Durchschnitt mehr Aussagen in der Kategorie ‚Bestätigung' aufweisen als in der Kategorie ‚Ansprache'. Das Bestätigen scheint für die Spieler somit von größerer Bedeutung zu sein als die direkte Ansprache anderer Gruppenmitglieder. Bei den Nichtspielergruppen gibt es hier kaum eine Tendenz. Der in allen Gruppen relativ niedrige Anteil an Aussagen zur Kommunikationssteuerung kann verschiedene Gründe haben: Entweder die Kommunikation verlief derart, dass explizite Aussagen zur Steuerung nicht notwendig waren, oder die Kommunikanten nutzten diese Möglichkeit nicht – obwohl notwendig. Setzt man diese Ergebnisse in Zusammenhang zu den Ergebnissen zur Kohärenz, ist anzunehmen, dass die Spieler nicht auf explizite Steuerung angewiesen waren, da im Grunde keine Kohärenzprobleme auftraten. Bei den Nichtspielern dagegen zeigten sich einige Schwierigkeiten bei der Aufrechterhaltung der Kohärenz, die Eingriffe in Form von steuernden Aussagen erfordert hätten.

Aufgabenorientierte Koordination

Zur Erfassung aufgabenorientierter Koordination dienen die Kategorien ‚Zielorientierung' und ‚Erfolgskontrolle'. Eine zusätzliche Informationsquelle stellt die Kategorie ‚Inhalt' dar, die zur Beschreibung der rein inhaltlichen Aufgabenorientierung dient. Abb. 6 stellt die relativen Häufigkeiten bezüglich dieser Kategorien dar.

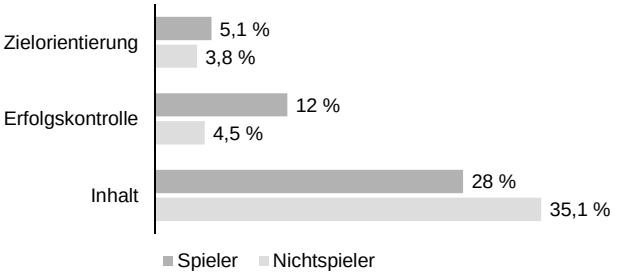


Abb. 6: Aufgabenorientierte Koordination

In der Kategorie ‚Zielorientierung' unterscheiden sich die Spielergruppen kaum von den Nichtspielergruppen. In der Kategorie ‚Erfolgskontrolle' dagegen weisen die Spielergruppen einen deutlich höheren Anteil an Aussagen auf als die Nichtspielergruppen (12% gegen 4,5%). Der niedrige Anteil der Nichtspielergruppen verweist darauf, dass nur wenige Aussagen getätigt wurden, die der Überprüfung der angeführten Inhalte oder Vorschläge dienten. Der vergleichsweise hohe Anteil bei den Spielergruppen weist dagegen darauf hin, dass in diesen Gruppen begleitende Überprüfungen beständiger Teil des Arbeitsprozesses waren. In der zusätzlichen Kategorie ‚Inhalt' ist die Höhe der Anteile an entsprechenden Aussagen bei den Nichtspielergruppen recht ähnlich; durchschnittlich beinhaltet ungefähr ein Drittel der Aussagen Ausführungen auf rein inhaltlicher Ebene. Also kann davon ausgegangen werden, dass die inhaltliche Aufgabenorientierung der Gruppen gut ausgeprägt war und das Gespräch nicht dauerhaft vom Thema der Aufgabe abschweifte.

Koordination der kooperativen Arbeit

Zur Beurteilung der Anstrengungen zur Verwirklichung von Kooperation und Gemeinschaftlichkeit, sind die Aussagen in den Kategorien ‚Bearbeitungsablauf', ‚Beitragssicherung', ‚Beteiligungssicherstellung' und ‚Zustimmung' von Interesse. Die relativen Häufigkeiten bezüglich dieser Kategorien sind in Abb. 7 dargestellt.

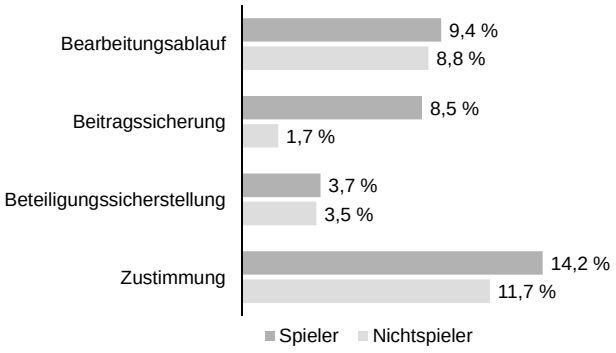


Abb. 7: Koordination der kooperativen Arbeit

In der Kategorie ‚Bearbeitungsablauf' unterscheiden sich die Spielergruppen im Durchschnitt der prozentualen Anteile kaum von den Nichtspielergruppen (9,4% gegen 8,8%). In der Kategorie ‚Beitragssicherung' übersteigen die Anteile entsprechender Aussagen in den Spielergruppen die Anteile in den Nichtspielergruppen dagegen deutlich (8,5% gegen 1,7%). Die hohen Anteile in den Spielergruppen weisen darauf hin, dass das Zusammenfassen von (Zwischen-)Ergebnissen für die Spieler ein fortlaufender Bestandteil des Arbeitsprozesses war. In der Kategorie ‚Beteiligungssicherstellung' wiederum weisen die Gruppen nur geringfügige Unterschiede auf (Spieler: 3,7%, Nichtspieler: 3,5%). In der Kategorie ‚Zustimmung' ist bezüglich der durchschnittlichen Werte zwischen den Spielergruppen und den Nichtspielergruppen ein Unterschied feststellbar (Spieler: 14,4%; Nichtspieler: 11,7%). Jedoch differieren die individuellen Werte innerhalb der Spielergruppen und innerhalb der Nichtspielergruppen so stark, dass diese Mittelwerte wenig aussagekräftig sind. Besonders auffällig ist der geringe Anteil an Aussagen einer der Nichtspielergruppen (4,1%). Während in den anderen Gruppen deutlich häufiger Zustimmung auf inhaltlicher Ebene geäußert wurde, was darauf hinweist, dass innerhalb dieser Gruppen eine recht hohe Einigkeit herrschte, scheint in dieser Gruppe eher keine Einigkeit zu herrschen. Dies kann z.B. daran liegen, dass Vorschläge aufgrund der zuvor erwähnten Kohärenzprobleme gar nicht verstanden wurden.

Die Erfassung doppelter Bestätigungen/Zustimmungen ermöglicht in diesem Zusammenhang weitere Aussagen. Abb. 8 stellt die Anteile doppelter Bestätigungen/Zustimmungen und die Anteile einzelner Bestätigungen und einzelner Zustimmungen bezogen auf die Gesamtanzahl aller Aussagen in den Kategorien ‚Zustimmung' und ‚Bestätigung' dar.

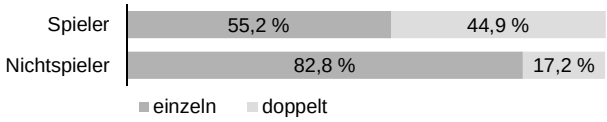


Abb. 8: Doppelte Bestätigungen/Zustimmungen

Bezüglich der doppelten Bestätigung ist zwischen den Spielergruppen und Nichtspielergruppen ein markanter Unterschied festzustellen. In den Spielergruppen wurden deutlich mehr Bestätigungen und Zustimmungen im Zusammenhang mit einer doppelten Bestätigung/Zustimmung gemacht als in den Nichtspielergruppen (Spieler: 44,9%; Nichtspieler: 17,2%). Dies unterstützt die vorangegangen Ausführungen bezüglich der Einigkeit in den Gruppen und kann als Hinweis darauf betrachtet werden, dass die Mitglieder der Spielergruppen Lösungen erarbeiteten, die tatsächlich auf der Zustimmung und dem Verständnis aller drei Beteiligten basierten, während die Mitglieder der Nichtspielergruppen eher abwechselnd bestätigten und so nicht erkennbar ist, ob das jeweils dritte Mitglied ebenfalls einverstanden war bzw. eine Äußerung überhaupt wahrgenommen und verstanden hat.

Anmerkung zu paralinguistischen Ausdrucksformen

Bezüglich des Einsatzes paralinguistischer Ausdrucksformen ist Folgendes festzuhalten: Die Spielergruppen nutzten deutlich mehr entsprechende

Ausdrücke als die Nichtspielergruppen, die insgesamt nur drei solche Ausdrücke zu verzeichnen hatten. Die Spieler nutzten offensichtlich die paralinguistischen Ausdrücke insbesondere zum Ausdruck oder zur Unterstützung des Gesprächsklimas sowie zur einfachen Bestätigung. Dies ist mit gängigem Verhalten in Face-to-Face-Situationen vergleichbar: Nonverbale Signale dienen häufig der Modulation des Klimas oder der Bestätigung im Sinne einer Demonstration von Aufmerksamkeit. Die Nichtspielergruppen dagegen nutzten diese Ausdrucksmöglichkeiten fast nicht. Dies kann einerseits damit erklärt werden, dass das entsprechende Wissen um den Einsatz solcher Ausdrucksformen bei den Nichtspielern nicht vorhanden oder gering ausgeprägt war. Andererseits mag dies mit einem mangelnden Bewusstsein darüber zusammenhängen, dass über ein Medium kommuniziert wird, das die Verschriftlichung nonverbaler Signale erfordert, sodass die Wirkung entsprechender Ausdrücke nicht erkannt wurde. So mag beispielsweise nicht erkannt worden sein, dass fehlende nonverbale Signale in Kombination mit einer verkürzten Ausdruckweise eine von der Intention des Sprechers abweichende Interpretation seitens der Adressaten begünstigen.

Arbeitsprozess

Wie bereits angemerkt, deuten die Ergebnisse auf Unterschiede im Arbeitsprozess zwischen den Spielergruppen und den Nichtspielergruppen hin. Insbesondere in den Kategorien ‚Erfolgskontrolle' und ‚Beitragssicherung' weisen die Spielergruppen deutlich höhere Anteile an Aussagen auf. Typische Elemente des Arbeitsprozesses in den Spielergruppen scheinen also das begleitende Überprüfen und Kontrollieren von Angaben sowie wiederholtes Festhalten von (Zwischen-) Ergebnissen zu sein. Bezüglich des Prozessergebnisses kann aufgrund der hohen Anteile an doppelten Bestätigungen/Zustimmungen zudem davon ausgegangen werden, dass die Lösungen der Spielergruppen auf der Zustimmung und dem Verständnis aller drei Beteiligten basieren.

Um weitere Unterschiede zu verdeutlichen, soll im Folgenden kurz – im Sinne einer qualitativen Auseinandersetzung mit dem Material – der Verlauf der Gruppenarbeiten dargestellt werden: Beide Spielergruppen bestimmten zu Anfang einen Schriftführer, der das Gruppenergebnis in einem externen Dokument festhalten sollte. Die Gruppen erarbeiteten die Lösungen schrittweise unter Einbezug aller Gruppenmitglieder. Prozessbegleitend wurden immer wieder Überprüfungen und Zusammenfassungen von Zwischenergebnissen vorgenommen. Zudem ist deutlich, dass alle Gruppenmitglieder den Stand der endgültigen Gruppenlösung kennen, der als Endergebnis abgegeben wurde. Das jeweilige Endergebnis in dem externen Dokument entsprach bei beiden Gruppen den Ausführungen im Chat.

Der Verlauf der Gruppenarbeit in den beiden Nichtspielergruppen war dagegen sehr unterschiedlich. Nur eine Nichtspielergruppe bestimmte anfangs einen Schriftführer. Diese Gruppe kam recht schnell zu einer Lösung, die allerdings – nach einer etwas längeren und stillen Denkpause – auf dem Komplettlösungsvorschlag eines Gruppenmitgliedes basierte. Das Endergebnis dieser Gruppe wurde anscheinend von allen so verstanden und akzeptiert; auch die Darstellung des Ergebnisses in dem externen Dokument entsprach den Ausführungen im Chat. In der anderen Nichtspielergruppe dagegen waren die anfänglichen Aussagen zur Bestimmung eines Schriftführers nicht eindeutig. Auch in dieser Gruppe gab es eine einzelne Person, die viele inhaltliche Aussagen und Teillösungsvorschläge einbrachte. Diese waren für die anderen Gruppenmitglieder anscheinend aber nicht nachvollziehbar, da wiederholt Bemerkungen gemacht wurden, die das jeweilige Unverständnis ausdrückten. Diese einzelne Person gestaltete letztendlich die Lösung alleine und gab sie als Endergebnis der Gruppe in einem externen Dokument ab. Die anderen beiden Gruppenmitglieder schienen nicht zu wissen, wie diese Lösung aussah. Die Darstellung der Lösung in dem externen Dokument ist beim Vergleich mit dem entsprechenden Chat-Protokoll nicht nachvollziehbar, da sie von den Ausführungen im Chat stark abweicht.

Die Ausführungen zeigen, dass die Arbeitsweise der Spielergruppen stark von der Arbeitsweise der Nichtspielergruppen abweicht. Interessant ist insbesondere, dass die beiden Spielergruppen unabhängig voneinander eine sehr ähnliche Verfahrensweise wählten.

Diskussion der Ergebnisse im Hinblick auf Adaptionspotenzial für Softwareentwicklungsteams

Ausgangspunkt der Überlegungen war die Fragestellung, wie sich die Koordination als für agile verteilte Softwareentwicklung wichtiger Kompetenzbereich strukturieren lässt, um sie in der Software-Engineering-Ausbildung lehren zu können. Dabei stand die Frage nach den Elementen der Koordinationskompetenz und möglichen Unterschieden bei Studierenden mit und ohne Erfahrung mit Online-Rollenspielen im Vordergrund.

Hinsichtlich der Elemente der Koordinationskompetenz bei der verteilten kooperativen Aufgabenbearbeitung mit zeitlichen Restriktionen zeigte sich, dass folgende kommunikative Praktiken zielführend und hilfreich waren:

Eine direkte Ansprache der Teammitglieder (z.B. durch Namensnennung) ist insbesondere bei Kohärenzproblemen hilfreich, um diesen entgegenzuwirken. Ein entsprechendes Problem muss allerdings zunächst als solches von den Teilnehmenden erkannt werden. Für die Vermittlung koordinativer Kompetenzen ist eine Sensibilisierung der Studierenden für diese Problematik sinnvoll. Sie sollten lernen, dass sie neben der eigenen Teilnahme auch genau auf den Verlauf achten müssen und unzusammenhängende Aussagen direkt ansprechen und klären sollten.

Eine regelmäßige Verbalisierung von Zwischenergebnissen ist wichtig für den Verlauf der Diskussion. Dies gewährleistet, dass alle Teilnehmenden auf demselben Stand sind und ermöglicht zugleich einen Abgleich bzw. eine Überprüfung des Standes. Dies ist auch eine Gelegenheit, Missverständnisse zu erkennen und zu klären.

Die Wahrnehmung von Vorschlägen und Aussagen sollte textuell kenntlich gemacht werden. Da der Körper als Zeichenträger fehlt, kann beispielsweise Aufmerksamkeit nicht durch ein Nicken o.ä. ausgedrückt werden. Es ist wichtig, Studierenden dies bewusst zu machen, insbesondere wenn keine oder kaum Vorerfahrungen vorhanden sind. Entsprechend sollte Aussagen und Vorschlägen von allen Beteiligten explizit zugestimmt oder diese explizit abgelehnt werden; das zeigt sich beispielsweise an der Bedeutung und Nutzung doppelter Zustimmungen in der Spielergruppen der Untersuchung. Nur so kann sichergestellt werden, dass tatsächlich alle Teilnehmenden beteiligt sind und z.B. eine Entscheidung mittragen. Auch zeigen mehrfache Bestätigungen, dass individuelle Bereitschaft der einzelnen Teilnehmenden da ist, den Arbeitsprozess mitzugestalten und Verantwortung zu übernehmen. Zugleich stärkt diese Praktik die Kohärenz des Gesprächs und trägt zur Klärung des aktuellen Stands in der Gruppe bei.

In der Lehre gilt es hier, dieses Wissen nicht nur kognitiv zu vermitteln, sondern entsprechende Lernbühnen zu schaffen, um diese Notwendigkeit zur Verbalisierung erfahrbar zu machen und zu motivieren (Herzberg & Marsden, 2005; Marsden, 2008, 2009; Marsden & Connolly, 2010). So kann – und sollte – beispielsweise das Wissen dahingehend, dass in der medial vermittelten Kommunikation die Verschriftlichung nonverbaler Signale eine wichtige Kommunikationspraktik ist, auch als theoretisch Wissen vermittelt werden. Doch nur indem dieses Wissen um eine Kommunikationspraktik in einer Situation praktisch geübt und idealerweise emotional verankert wird, kann sie den Studierenden nachhaltig vermittelt werden.

Hier ist es wichtig, in der Lehre im Software Engineering solche verteilte, medial vermittelte Teamarbeiten notenrelevant zu verankern und im Hinblick auf erfolgreiche Kommunikationspraktiken zu analysieren. Dabei ist von Bedeutung, dass es sich tatsächlich um verteilt arbeitende Teams handelt, beispielsweise durch die Zusammenarbeit mehrerer Hochschulen. Eine Simulation – beispielsweise durch bloßes Verteilen auf verschiedene Räume – schmälert die Lernerfahrung erheblich (Kasparek & Marsden, 2007). Hilfreich könnte es auch sein, bei der Zusammensetzung der Teams darauf zu achten, dass Diversität in den Teams auch hinsichtlich der Erfahrung mit Online-Rollenspielen gegeben ist. Im Idealfall können diese Vorerfahrungen dazu führen, dass durch das Voneinanderlernen der Studierenden mit unterschiedlichem Kompetenzstand innerhalb der Gruppen der Lernprozess beschleunigt wird.

Bezogen auf die Unterschiede durch vorherige Erfahrungen mit synchroner textbasierter Kommunikation im Kontext von Online-Rollenspielen hinsichtlich der Koordinationsleistung zeigte sich, dass die Studierenden mit entsprechender Vorerfahrung mit ihren Kommunikationsweisen insgesamt erfolgreicher waren als die Studierenden, die keine Erfahrung mit Online-Rollenspielen hatten. Im Kontext der Lehre von virtueller Zusammenarbeit ist es ausgesprochen instruktiv, dass die spielerfahrenen Studierenden im Rahmen der Aufgabenbearbeitung rein quantitativ deutlich mehr Aussagen machen als nicht-spielerfahrene Studierende. Dieses Ergebnis deckt sich mit Beobachtungen aus Lehrveranstaltungen zur computervermittelten Kommunikation und zur verteilten Zusammenarbeit im Studiengang Software Engineering an der Hochschule Heilbronn (Kasparek & Marsden, 2007; Marsden, 2012a, 2012b, 2013a, 2013b): Hier hat sich in der qualitativen Analyse der Logfiles der Beiträge der Studierenden zur Benotung immer wieder gezeigt, dass es eine hohe Korrelation der Quantität der Meldungen mit der Qualität des Beitrages der einzelnen Studentin bzw. des Studenten gibt: Je aktiver sich die Studierenden an der textbasierten synchronen Kommunikation beteiligen, umso konstruktiver ist ihr Beitrag zur Koordination der Gruppe und zum Gruppenergebnis.

Fazit

Angesichts des Erfolgskurses der agilen Programmiermethoden auf der einen und der Globalisierung der Softwareentwicklung auf der anderen Seite ist es wichtiger als je zuvor, den Aufbau von Kompetenzen zur Kommunikation und Koordination im Software Engineering praxisnah und berufsbefähigend zu vermitteln.

Die hier untersuchten Kommunikationspraktiken zur Koordination von Kommunikationsprozessen in der medial vermittelten Zusammenarbeit stellen grundlegende Voraussetzungen dar, um

verteilte Zusammenarbeit im Team erfolgreich zu bewältigen. Sie sind für die Praxis der verteilten Softwareentwicklung nicht zuletzt deshalb von Interesse, da viele Probleme und Prozessverluste im Rahmen von medial vermittelter Zusammenarbeit mit mangelnden (tele-)kommunikativen Fähigkeiten zu erklären sind. Dabei ist eine Adaption an das jeweilige Kommunikationsmedium sowohl für den Umgang mit diesem Medium als auch für die Bereitschaft, ein Medium zu nutzen, von Bedeutung.

Unterstrichen wird die Bedeutsamkeit dieser Kompetenzen dadurch, dass in der agilen Softwareentwicklung die Selbstorganisation in Teams elementarer Bestandteil der Herangehensweise ist (Meyer, 2014) und ein konstruktives Miteinander somit entscheidend für die Qualität des Ergebnisses ist. Vor diesem Hintergrund geben die agilen Methoden Strukturierungshilfen für Gruppenprozesse, wie z. B. das „Planning Game" im XP oder das „Planning Poker" im Scrum, in dem das Team sich auf eine Schätzung einigt, indem jede Person zunächst eine Schätzung anhand vorgegebener Aufwandskategorien (Zahlen der Fibonacci-Reihe) festhält um dann auf dieser Basis weiter zu diskutieren. Die Tatsache, dass hier entsprechende Hilfestellungen zur Strukturierung des Gruppenprozesses entwickelt bzw. mitgegeben werden müssen, zeigt jedoch auch, dass systematische Methoden zur Strukturierung und Koordination der Kommunikation in den verschiedenen Phasen der Teamarbeit in der Softwareentwicklung bisher wenig etabliert sind. Da die Vermittlung dieser Methoden schon für die Face-to-Face-Interaktion außerordentlich komplex und aufwändig ist (Herzberg & Marsden, 2005; Marsden, Herzberg & Drescher, 2008), ist es nachvollziehbar, dass im Sinne des agilen Manifests hier eine Beschränkung auf das „Gespräch von Angesicht zu Angesicht" (Beck et al. 2001) wünschenswert ist. Die in diesem Beitrag vorgestellte Untersuchung unterstreicht dies, indem sie aufzeigt, dass die Übertragung der face-to-face vorhandenen Kompetenzen zur wirksamen Koordination von Zusammenarbeit im Team auf die medial vermittelte Kommunikation schwierig ist. Sie zeigt allerdings auch, dass es eine durch Übung erlernbare Kompetenz ist. Angesichts der Tatsache, dass in Zukunft nicht nur die Verbreitung der agilen Methoden, sondern auch die verteilte Zusammenarbeit in der Softwareentwicklung weiter zunehmen wird, scheint es wichtig, dass die Kompetenz zur Steuerung von Teamprozessen in der medial vermittelten Kommunikation im Software-Engineering-Unterricht der Hochschulen nachhaltig verankert wird.

Literatur

Alzoubi, Y., Gill, A. (2014). Agile global software development communication challenges: A systematic review. In *Pacific Asia Conference on Information Systems (PACIS 2014)*. Chengdu, China, No 20.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). Agile Manifesto. Abgerufen am 31. Oktober 2014 von http://agilemanifesto.org/

Boden, A. (2011). Coordination and learning in global software development: articulation work in distributed cooperation of small companies. Dissertation, Universität Siegen.

Böttcher, A., Thurner, V., Müller, G. (2011). Kompetenzorientierte Lehre im Software Engineering. In J. Ludewig & A. Böttcher (Hrsg.), Software Engineering im Unterricht der Hochschulen (SEUH 2011). München, S. 33-39.

Carrillo de Gea, J.M., Nicolás, J., Fernández Alemán, J.L., Toval, A., Vizcaíno, A., Ebert, C. (2013). Reusing Requirements in Global Software Engineering. In W. Maalej & A.K. Thurimella (Hrsg.), Managing Requirements Knowledge. Berlin, Heidelberg: Springer-Verlag, S. 171-197.

Clark, H., Brennan, S. (1991). Grounding in communication. *Perspectives on socially shared cognition*, 13 (1991), S. 127-149.

Eckstein, J. (2013). Agile software development with distributed teams: Staying agile in a global world. Addison-Wesley.

Hanssen, G. K., Smite, D., Moe, N. B. (2011). Signs of agile trends in global software engineering research: A tertiary study. In *Sixth IEEE International Conference on Global Software Engineering Workshop (ICGSEW)*. Helsinki, Finnland, S. 17-23.

Herzberg, D., Marsden, N. (2005). Das Softwarelabor als Lernbühne - Soziale Kompetenzen im Studiengang Software Engineering praxisnah vermitteln. In B. Berendt, H. Voss & J. Wildt (Hrsg.), Neues Handbuch Hochschullehre. Lehren und Lernen effizient gestalten. Berlin: Raabe, G.5.3.

Hossain, E., Bannerman, P. L, Jeffery, D. R. (2011). Scrum practices in global software development: a research framework Product-focused software process improvement. Berlin, Heidelberg: Springer-Verlag, S. 88-102.

Kasparek, M., Marsden, N. (2007). Einfluss von Qualitätsdruck und Kontinuität der Zusammenarbeit auf virtuelle Teamarbeit. In A. Zeller & M. Deininger (Hrsg.), Software Engineering

im Unterricht der Hochschulen (SEUH 10). Stuttgart, S. 83-86.

Larman, C., Vodde, B. (2010). Practices for scaling lean and agile development: large, multisite, and offshore product development with large-scale Scrum. Boston: Pearson Education.

Malone, T.W., Crowston, K. (1990). What is coordination theory and how can it help design cooperative work systems. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work (CSCW '90)*. New York, NY, USA, S. 357-370.

Marsden, N. (2008). Motivation durch Instruktion? Effekte der Induktion von Performanzzielorientierung. *Zeitschrift für E-Learning*, 1(3), S. 32-44.

Marsden, N. (2009). Lehr-Muster bei geringer Ausgangsmotivation für überfachliche Qualifizierung. In U. Jaeger & K. Schneider (Hrsg.), Software Engineering im Unterricht der Hochschulen (SEUH 11). Hannover, S. 101-110.

Marsden, N. (2012a). Subjektive Theorien als Startpunkt für die Lehre psychologischer Themen. In M. Krämer, S. Dutke & J. Barenberg (Hrsg.), Psychologiedidaktik und Evaluation IX. Aachen: Shaker Verlag, S. 189-196.

Marsden, N. (2012b). Teaching Hyperpersonal On-Line Collaboration. Paper presented at the *12th International Conference on e-Learning, e-Business, Enterprise Information Systems and e-Government (EEE'12)*. Las Vegas, USA.

Marsden, N. (2013a). Attitudes towards Online Collaboration: An Exploratory Factor Analysis. In *Proceedings of the 2013 annual conference on Computers and people research (SIGMIS-CPR '13)*. New York, NY, USA, S. 147-152.

Marsden, N. (2013b). Learning Online: The Role of Attitudes Towards Online Communication. Paper presented at the *13th International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government (EEE'13)*. Las Vegas, USA.

Marsden, N., Connolly, C. (2010). Pedagogical Patterns for Computer-Mediated Communication. In M. B. Nunes & M. McPherson (Hrsg.), Proceedings of the IADIS International Conference e-Learning 2010 (Vol. II). Freiburg, S. 27-32.

Marsden, N., Herzberg, D., Drescher, F. (2008). Zur Didaktik von Software-Engineering-Praktika aus Sicht der Studierenden. In: Forschungsbericht 2008 der Hochschule Heilbronn. Heilbronn: Institut für angewandte Forschung, S. 123-126.

Meyer, B. (2014). Agile! The Good, the Hype and the Ugly. Switzerland: Springer International Publishing.

Paasivaara, M., Lassenius, C., Damian, D., Raty, P., Schröter, A. (2013). Teaching students global software engineering skills using distributed scrum. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. Piscataway, NJ, USA, S. 1128-1137.

Paechter, M. (2003). Wissenskommunikation, Kooperation und Lernen in virtuellen Gruppen. Lengerich: Pabst.

Pries-Heje, L., Pries-Heje, J. (2011). Why Scrum works: A case study from an agile distributed project in Denmark and India. Paper presented at the *Agile Conference (AGILE)*. Salt Lake City, UT, USA.

Schneider, S., Torkar, R., Gorschek, T. (2013). Solutions in global software engineering: A systematic literature review. *International Journal of Information Management*, 33(1), S. 119-132.

Strode, D. E, Huff, S. L, Hope, B., Link, S. (2012). Coordination in co-located agile software development projects. *Journal of Systems and Software*, 85(6), S. 1222-1238.

Sutherland, J., Viktorov, A., Blount, J., Puntikov, N. (2007). Distributed scrum: Agile project management with outsourced development teams. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS '07)*. Washington, DC, USA, No 274a.

Walther, J. B. (2012). Interaction Through Technological Lenses: Computer-Mediated Communication and Language. *Journal of Language and Social Psychology*, 31(4), S. 397-414.

Williams, L. (2010). Agile software development methodologies and practices. In M.V. Zelkowitz (Hrsg.), Advances in Computers. Amsterdam: Elsevier, S. 1-44.

Mit Projekten ins Studium starten

Karin Vosseberg, Hochschule Bremerhaven

kvosseberg@hs-bremerhaven.de

Zusammenfassung

Für das Fach Software Engineering ist eine der großen Herausforderung den Studierenden, die nicht über Erfahrungen großer (gescheiterter) Softwareentwicklungprojekte verfügen, die Notwendigkeit eines methodischen Vorgehens zu vermitteln. Mit einem Studieneinstiegsprojekt werden die Studierenden der Bachelorstudiengänge Informatik und Wirtschaftsinformatik an der Hochschule Bremerhaven auf die komplexen Anforderungen in einem Softwareentwicklungsprozess vorbereitet.

Motivation

Viele der Studienanfänger und Studienanfängerinnen haben kaum eine Vorstellung von den komplexen Aufgaben in der Informatik und Wirtschaftsinformatik. Nach wie vor ist das Bild geprägt durch viele Mythen. Während in der Informatik immer noch das Klischee der Kellerkinder vorherrscht, die sich in die programmiertechnischen Herausforderungen eingraben, glauben die Studierenden der Wirtschaftsinformatik, dass sie als zukünftige Projektmanager keine fundierten Kenntnisse in der Softwareentwicklung selber benötigen. Beide Sichtweisen macht es für ein Fach wie Software Engineering schwer ein methodisches Vorgehen in der Softwareentwicklung zu motivieren.

Um das Bild der Informatik und Wirtschaftsinformatik für Studienanfängerinnen und Studienanfänger nachhaltig zu schärfen, starten wir seit dem WS 2013/14 das Studium mit einem siebenwöchigen Studieneinstiegsprojekt. In diesen Projekten analysieren die Studierenden ein vorgegebenes komplexes Anwendungsumfeld und erarbeiten für einen Teilbereich den Einsatz von IT-Systemen zur Unterstützung der Geschäftsprozesse (Vosseberg et al. 2014). In diesem Jahr steht eine fiktive Schokoladenfabrik[1] im Fokus der Studieneinstiegsprojekte.

Rahmenbedingungen

Bereits am ersten Studientag bilden wir 12 studiengangsübergreifende Teams mit jeweils 6-8 Studierenden. Jeweils drei Teams werden von einer oder einem Lehrenden als Coach betreut. Zusätzlich wird jedem Team ein Tutor oder eine Tutorin aus einem höheren Semester zugeordnet.

In den ersten sieben Wochen des ersten Semesters werden die Modulstrukturen weitestgehend aufgebrochen. Die Grundlagenveranstaltungen wie Programmieren I und Mathematik I laufen parallel zum Studieneinstiegsprojekt. Der notwendige Arbeitsaufwand für diese Fächer, ist in die wöchentliche Arbeitszeit der Teams eingebunden, so dass die Teams gemeinsam die notwendigen Übungen bearbeiten können. Die weiteren Grundlagenveranstaltungen aus dem ersten Semester, wie Einführung in die Informatik bzw. Wirtschaftsinformatik, Software Engineering I – Modellierung, Einführung in die BWL und wissenschaftliches Arbeiten liefern mit ihren Inhalten erste Grundlagen für die Bearbeitung der Projekte. Die Lehrenden aus den entsprechenden Veranstaltungen unterstützen die Studierenden in ihrem problemorientierten Lernansatz und liefern den notwendigen Input auf die Projekte bezogen (vgl. Webler 2012) .

Ab der 8. Woche beginnt der „normale“ Veranstaltungsbetrieb für alle Module. Das Studieneinstiegsprojekt ist formal dem Modul Einführung in Informatik bzw. Wirtschaftsinformatik mit 5CP zugeordnet. Die Ergebnisse des Studieneinstiegsprojekts werden im Rahmen dieses Moduls von den Coaches bewertet. Die Inhalte und erworbenen Kompetenzen der anderen Module werden unabhängig vom Studieneinstiegsprojekt bewertet.

Studieneintiegsprojekte

Den Teams wird für ihr Studieneinstiegsprojekt ein breitgefächertes Problemumfeld – die innovative Modernisierung einer fiktiven Schokoladenfabrik – vorgegeben. Jedes Team analysiert die Schokoladenfabrik und bewirbt sich für eine Abteilung, um für diese Abteilung konkrete Ideen zur Unterstützung der Geschäftsprozesse durch IT-Systeme zu entwickeln. Außerdem sollen sie fehlende Grundlagen identifizieren, die für eine professionelle Lösung der entwickelten Ideen erforderlich sind.

Als Input erhalten die Teams neben der Beschreibung der fiktiven Schokoladenfabrik eine Exkursion in ein konkretes Unternehmen – entweder in ein produzierendes Unternehmen ähnlich der Schokoladenfabrik oder in ein Softwareberatungsunternehmen. Ihre Aufgabe ist, die Erfahrun-

[1] ähnlich der Beschreibung unter:
sites.google.com/site/xocschokolade/home;
Stand 01.11.2014

gen aus den Exkursionen auf die fiktive Situation der Schokoladenfabrik zu übertragen. Dabei werden sie durch die Lehrenden aus dem ersten Semester (BWL, Software Engineering I und Wissenschaftliches Arbeiten) unterstützt.

Nach 3 Wochen präsentieren die Teams ihre ersten Ergebnisse insbesondere die Erfahrungen aus den Exkursionen im Plenum. Durch die 12 sehr unterschiedlichen Unternehmen, die die Teams in ihren Exkursionen besucht haben, wird in diesen Vorträgen ein vielfältiges Aufgabenspektrum der Informatik und Wirtschaftsinformatik aufgeblättert. Nach 7 Wochen werden die Endergebnisse von den Teams wiederum im Plenum präsentiert.

Einstieg für Software Engineering

Im Modul Software Engineering I liegt bei uns der inhaltliche Schwerpunkt auf dem Thema Modellierung und das Erlernen einer Modellierungssprache. In den ersten Wochen der Studieneingangsphase wird mit den Teams gemeinsam diskutiert, wie sie Modelle in der Analyse ihrer Schokoladenfabrik gewinnbringend einsetzen können. Sie müssen erste Modelle ihrer Schokoladenfabrik erarbeiten und vorstellen ohne dass sie bereits eine konkrete Modellierungssprache nutzen.

In der weiteren Entwicklung ihrer Lösungsansätze für die einzelnen Abteilungen der Schokoladenfabrik erarbeiten sich dann die Teams erste UML-Diagramme selber. Beispielsweise nutzen sie Aktivitätsdiagramme zur Beschreibung von Abläufen. Entlang dieser Diagramme wird die Notwendigkeit von Syntax und Semantik von Modellierungssprachen besprochen. Außerdem wird an den konkreten Modellbeispielen diskutiert wie diese genutzt werden können, um zu einer konkreten Softwarelösung zu kommen. Damit wird für die Studierenden immer wieder der Bezug zur Programmierausbildung hergestellt. Sie erfahren anhand ihres konkreten Beispiels, dass zur Softwareentwicklung mehr gehört als die programmiertechnische Umsetzung einzelner Algorithmen.

Die Teams müssen immer wieder ihre erarbeiteten Modelle zur Diskussion stellen und lernen damit Modelle als Kommunikationselement einzusetzen um eine gemeinsame Lösung zu erarbeiten. Im Anschluss an die Studieneingangsphase werden die Ergebnisse in der Veranstaltung Software Engineering I aufgegriffen und die verschiedene Diagrammtypen von UML immer wieder auf die Beispiele aus der Schokoladenfabrik bezogen.

Erfahrungen

Geprägt durch ihre Erfahrungen aus dem schulischen Bereich haben viele Studierende große Schwierigkeiten mit der offenen Fragestellung. Sie sind gewohnt, dass Aufgabenstellungen detailliert heruntergebrochen werden und anschließend eine korrekte Musterlösung zur Verfügung steht. Sie müssen erst an die veränderte Situation und die an sie gestellten Anforderungen herangeführt werden.

Hinzu kommt die Schwierigkeit in Teams zusammenzuarbeiten. Alle Teams haben häufig zum ersten Mal Erfahrungen mit Projektmanagementaufgaben gemacht. Neben wöchentlichen Zeitplänen mussten die Teams Aufgabenpakete definieren und grobe Zeitschätzungen des Aufwands abgeben und ihren realen Aufwand erfassen.

Viele Teams haben gut zu einander gefunden und sich als Lerngruppe sehr gut gegenseitig unterstützt, während andere Teams mehr eine „Zwangsgemeinschaft“ sind, die die Studieneingangsphase gemeinsam durchstehen müssen. Die Coaches haben die Aufgabe, hier immer wieder steuernd einzugreifen. Zusätzlich ist die Unterstützung der Teams durch ältere Studierende gerade als erste Ansprechstation sehr hilfreich.

In der Veranstaltung Software Engineering I hilft das Einstiegsprojekt die Vorteile aus der Entwicklung von Modellen und die Nutzung einer Modellierungssprache zu erkennen. Die ersten Modelle waren zwar holprig aber zeigten schon gute Ansätze an denen sich die Diskussion über die Modelle entfachte.

Fazit

Studienprojekte als adäquates Mittel, um Studierende an die Methoden und Vorgehensweisen im Software Engineering heranzuführen, werden seit Beginn der SEUH[2]-Workshopreihe immer wieder sehr eindrucksvoll beschrieben und mit vielen Erfahrungen belegt. Auch für den Studieneinstieg zeigt sich, dass die Projekte die Motivation der Studierenden, sich mit Methoden des Software Engineering auseinander zusetzen, stark erhöht wird und sie vor ihrem Erfahrungshintergrund zu beachtlichen Leistungen kommen.

Literatur

Vosseberg, K.; Czernik, S.; Erb, U.; Vielhaber, M. (2014): Projektorientierte Studieneingangsphase: Das Berufsbild der Informatik und Wirtschaftsinformatik schärfen. In: Proc. 6. HDI, Freiburg, 09/2014, ISBN 978-3-86956-313-8

Webler, W.-D. (2012): Entwicklung des Erstsemesterprojekts an der Fakultät für Forst- und Umweltwissenschaften der Universität Freiburg. In: Webler, Wolff-Dietrich (Hrsg.): Studieneingangsphase? Das Bachelor-Studium braucht eine neue Studieneingangsphase! Bielefeld: Universitäts-Verl. Webler, S. 209-221

[2] www.seuh.org