Technology
Arts Sciences
TH Köln

Digitalisierung und Individualisierung der praxisorientierten Lehre von modernen Coding-Ansätzen

Masterarbeit zur Erlangung des Mastergrades
Master of Science im Studiengang Informatik Software Engineering
an der Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln

vorgelegt von:	Jann Intveen
Matrikel-Nr:	11106564
Adresse:	Aachener Str. 1360
	50859 Köln
	JannIntveen@gmail.com
eingereicht bei:	Prof. Dr. Stefan Bente
Zweitgutachter:	Prof. Dr. Christian Kohls

Köln, 17. April 2021

Abstract

In the context of this work, the practice-oriented teaching of modern programming approaches is examined from various perspectives. In addition to the literature, the findings of expert interviews are also used to make recommendations on the design of modern programming courses. Furthermore, it turned out that the programming approaches Clean Code, Domain Driven Design and Test Driven Development can be integrated into the teachings by considering certain guidelines and adapting associated requirements to the basic conditions of the teaching environment. Additionally, the relevance of frequent feedback for learners is emphasized and different feedback possibilities are discussed. To enable feedback on a large scale, appropriate tool support is needed. As a result, Divekit is presented as a tool that supports a Gitlab based automatic testing concept and also integrates functionalities for individualizing exercises in order to make it more difficult for learners to copy the solutions of others.

Inhaltsverzeichnis

Listings

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung

Um moderne Coding-Ansätze zu lehren und zu vermitteln, müssen diese von Lernenden im Rahmen von Praktika angewendet werden. Erst dann kann das ganze Ausmaß dieser Coding-Ansätze verstanden und verinnerlicht werden. Eine besondere Herausforderung stellt dabei das Ausrichten des Praktikums auf moderne Sprachen und Konzepte wie Domain Driven Design, Test Driven Development oder Clean Code dar.

Ein dementsprechendes Praktikum hat dabei zwei primäre Aufgaben. Auf der einen Seite soll es Lernenden ermöglicht werden, jene Inhalte, welche vorab vermittelt werden, anwenden zu können. Auf der anderen Seite soll der Erfolg eines jeden Lernenden in Bezug auf das Praktikum jederzeit nachvollziehbar und prüfbar sein. Zusammenfassend könnte von einer Digitalisierung des Praktikums gesprochen werden. Besonders die zur Zeit der Verfassung dieser Arbeit aktuelle Corona-Lage verleiht dem Faktor Digitalisierung besonders viel Gewicht.

Geschuldet der zeitlichen Überlastung und der arbeitsteiligen Selbstorganisation der Lernenden ist leider oft nicht nachvollziehbar, welcher Lernender eine Coding Aufgabe nun letztendlich gelöst hat. Häufig bilden sich Lerngruppen aus Lernenden, dessen einzige Aufgabe darin besteht, fertige Lösungen untereinander auszutauschen. Um dem entgegenzuwirken, kann ein bestimmter Grad an Individualisierung von Aufgaben Abhilfe schaffen.

Eine weitere Anforderung an diese Art der praxisorientierten Lehre wäre ein möglichst hoher Grad an Automatisierung, um die Ressource Personal und die damit verbundene Betreuung an den richtigen Stellen einsetzen zu können. Dadurch würde die Beratungsleistung hochwertiger ausfallen, weil Betreuer weniger Routineaufgaben zu erledigen hätten.

1.2 Ziel

Das Ziel dieser Arbeit lässt sich in zwei Bereiche aufteilen. Zum einen soll ein Konzept ermittelt werden, welches beschreibt, wie moderne Coding Ansätze am besten praxisorientiert vermittelt werden können. Es wird auch geklärt, welche Aspekte sich überhaupt modernem Coding zuordnen lassen. Außerdem gilt es herauszufinden, wie mithilfe von Digitalisierung Teile des Praktikums automatisiert werden können, um von den dadurch entstehenden Vorteilen zu profitieren. Ein wichtiger Bestandteil dieser Digitalisierung stellt außerdem das automatische Individualisieren von Praktikumsaufgaben dar, um dem Pro-

blem entgegenzuwirken, dass oftmals Lösungen von Aufgaben kopiert werden.

Ein weiteres Ziel dieser Arbeit ist das Zusammenstellen von geeigneten Werkzeugen, welche benötigt werden, um vorher beschriebenes Konzept umzusetzen. Es handelt sich hierbei um verschiedene Arten von Werkzeugen, wobei sich diese je einem anderen Anwendungszweck zuordnen lassen. Die Auswahl und Entwicklung dieser Werkzeuge orientieren sich dabei stark an realen Anforderungen aus der praxisorientierten Lehre. Diese Anforderungen basieren auf Literaturrecherchen und durchgeführten Experteninterviews.

1.3 Aufbau

Der erste Teil der Arbeit beschäftigt sich mit den Grundlagen und sich daraus ergebenden Anforderungen. Darunter fallen zum einen Konzepte der Didaktik und zum anderen moderne Konzepte aus der Programmierung. Zusätzlich werden die Ergebnisse von durchgeführten Experteninterviews genutzt, um aus der Literatur gewonnene Anforderungen anzureichern.

Im nächsten Schritt werden bereits existierende Lösungen vorgestellt, die im Problemstellungsbereich dieser Arbeit lokalisiert sind. Ausgewählt werden diese auf Basis der Anforderungen, welche im ersten Teil der Arbeit herausgearbeitet werden.

In dem darauffolgenden Kapitel werden Empfehlungen und Ansätze vorgestellt, welche beschreiben, wie Programmierpraktika gestaltet und organisiert werden können. Diese Empfehlungen und Ansätze basieren auf den Konzepten, welche in den Grundlagen vorgestellt werden und auf den Ergebnissen der Experteninterviews.

Anschließend werden auf Basis erarbeiteter Empfehlungen und Ansätze Werkzeuge zur Automatisierung und Individualisierung beschrieben, welche sich nach Anforderungen richten, die mit vorher beschriebenen Empfehlungen und Ansätzen einhergehen.

In einem letzten Kapitel werden drei Fallstudien vorgestellt, welche Erkenntnisse in Hinblick auf die Anwendung von thematisierten Konzepten und unterstützenden Werkzeugen in der Praxis liefern.

Den Abschluss der Arbeit bilden das Fazit und der Ausblick.

2 Grundlagen

Das Grundlagenkapitel dieser Arbeit lässt sich in zwei Teilbereiche unterteilen. Zum einen werden wichtige Konzepte der Didaktik vorgestellt, zum anderen werden moderne Konzepte der Programmierung thematisiert, welche in die praxisorientierte Lehre von modernen Programmieransätzen integriert werden können. Im Anschluss an jedes dieser beiden Kapitel folgt eine Anforderungsliste, dessen Anforderungen sich aus den jeweils vorhergehenden Kapiteln ergeben. Diese Anforderungen sind im späteren Verlauf dieser Arbeit relevant, denn diese beziehen sich auf die Gestaltung der praxisorientierten Lehre von modernen Programmieransätzen und auf unterstützende Werkzeuge.

2.1 Konzepte der Didaktik

Im Rahmen dieser Arbeit werden Empfehlungen für die Gestaltung von Programmierpraktika ausgesprochen. Diese Empfehlungen begründen sich zum Teil durch bestimmte Konzepte aus dem Bereich der Didaktik, welche im Folgenden thematisiert werden.

2.1.1 Active Learning

Wenn moderne Programmierung und dazugehörige Konzepte gelehrt werden sollen, dann müssen diese von den Lernenden auch praktisch angewendet werden, um den Lernerfolg zu erhöhen. Die Notwendigkeit von praktischer Anwendung ist natürlich auch in anderen Bereichen der Lehre vorhanden. Daher existieren diverse Ansätze, um diese praktische Anwendung zu gewährleisten. Einer dieser Ansätze nennt sich *Active Learning*. Im Rahmen dieses Ansatzes werden Lernende mehrmals hintereinander mit bestimmten Problemstellungen konfrontiert, welche in verschiedenen Aktivitäten münden. Dabei ist je nach Ziel sowohl Einzelarbeit als auch Gruppenarbeit möglich. Die Aufgabe von Lehrenden ist es, in bestimmten Intervallen Erkenntnisse zusammenzutragen, Inhaltsimpulse zu geben und neue Problemstellungen zu präsentieren. Einfaches Fragenstellen während eines Inhaltsimpulses sollte nicht Active Learning zugeordnet werden, denn bei diesem Ansatz sollten sich Lernende für die einzelnen Aktivitäten und die dazugehörige Problemstellung Zeit nehmen können. Dies gewährleistet, dass sich die Lernenden ausgiebig mit der Problemstellung beschäftigen können. Ziel sollte auch nicht sein, dass Lehrveranstaltungen ausschließlich aus beschriebenen Aktivitäten bestehen. Gezielte Inhaltsimpulse sind nötig, um Wissen zu vermitteln, welches während der Aktivitäten benötigt wird (vgl. Felder und Brent 2009, S. 1f).

Fragen oder Problemstellungen, welche Aktivitäten einleiten, sollten thematisch natürlich zu dem passen, was gerade gelehrt wurde. Außerdem sollte darauf geachtet werden, dass

Fragen nicht so einfach formuliert werden, dass diese schon von einem Großteil der Lernenden bereits zum Zeitpunkt der Fragestellung beantwortet werden können. Dies würde verhindern, dass sich ausgiebig mit der Problemstellung beschäftigt wird (vgl. Felder und Brent 2009, S. 2f).

Das Konzept Active Learning kann mithilfe eines Modells in vier Phasen untergliedert werden. Wichtig anzumerken ist dabei, dass diese Phasen je nach Lernziel oder Anwendungsbereich variieren können (vgl. Hazzan, Ragonis und Lapidot 2020, S. 25-27):

Erste Phase: Auslöser
In dieser ersten Phase werden die Lernenden mit einem Problem oder einer Fragestellung konfrontiert. Diese ist hinreichend komplex und beinhaltet viele gelehrte Aspekte, welche für die Problemlösung relevant sind.

Zweite Phase: Aktivität
Die Lernenden beschäftigen sich nun mit der vorliegenden Problemstellung. Die Dauer dieser Aktivitätsphase variiert stark je nach Art der Problemstellung und der gesetzten Lernziele.

Dritte Phase: Diskussion
Nach der Aktivitätsphase werden die verschiedenen Ergebnisse und Lösungen der Lernenden zusammengetragen und diskutiert. Hierbei reflektiert jeder seine eigene Lösung und trägt konstruktives Feedback zu den Lösungen anderer bei. Der Lehrende leitet die Diskussion und stellt relevante Merkmale heraus, versucht aber, verschiedene Lösungen nicht zu bewerten, denn dies geschieht über das konstruktive Feedback der anderen Lernenden. Allerdings wird durch den Lehrenden betont, dass verschiedene Lösungen für das Problem existieren.

Vierte Phase: Zusammenfassen
In dieser letzten Phase wird durch den Lehrenden auf die Zusammenhänge und die Relevanz der Thematik hingewiesen. Es werden zentrale Konzepte und Ideen zusammengefasst, welche sich aus den drei vorherigen Phasen ergeben haben.

2.1.2 Intrinsische und extrinsische Motivation

Um Lernfortschritt zu erzielen, müssen sich die Lernenden mit dem Lernstoff beschäftigen. Dazu ist ein gewissen Maß an Motivation nötig, es muss also Gründe für die Lernenden geben, sich mit dem Lernstoff auseinanderzusetzen. Wichtig dabei anzumerken ist, dass

sich verschiedene Menschen auf unterschiedliche Weise unterschiedlich stark motivieren lassen. Diese Möglichkeiten der Motivation lassen sich grob in zwei Kategorien unterteilen, die intrinsische und die extrinsische Motivation (vgl. Ryan und Deci 2000, S. 54f).

Intrinsische Motivation

Wenn man für eine Aktivität intrinsisch motiviert ist, bedeutet dies, dass man durch die Aktivität selbst motiviert wird. Dies kann verschiedene Gründe haben. Es könnte beispielsweise sein, dass man Spaß an der Aktivität hat oder dass man die Herausforderung sucht, welche die Aktivität birgt. Bei dieser Art der Motivation geht es also weniger um externe Faktoren wie künstlichen Druck oder Belohnungen. Deshalb ist diese Art der Motivation besonders wertvoll, da auf diese Weise angestoßene Lernprozesse besonders effektiv sind. Das Problem der intrinsischen Motivation liegt allerdings in dessen Natur selbst. Der Grad der intrinsischen Motivation hängt von dem jeweiligen Individuum und der betrachteten Aktivität ab (vgl. Ryan und Deci 2000, S. 56f). Daraus folgt, dass nicht jeder Lernende die gleiche Menge an intrinsischer Motivation für eine Aktivität mitbringt.

Extrinsische Motivation

Bei extrinsischer Motivation ist der Motivationstreiber nicht die Aktivität selbst, sondern ein davon separierbares Ziel. Ein solches Ziel könnte beispielsweise das Erhalten guter Noten sein. Man kann dabei zwischen vier Arten der extrinsischen Motivation unterscheiden, wobei anzumerken ist, dass verschiedene extrinsische Motivationstreiber unterschiedliche Auswirkungen auf individuelle Menschen haben (vgl. Ryan und Deci 2000, S. 60ff):

Externe Regulierungen: Die Motivation geschieht durch Belohnungen oder Bestrafungen.

Introjektion: Man wird durch das eigene Selbstgefühl motiviert, folglich will man sich selbst oder anderen etwas beweisen und sucht Bestätigung.

Identifikation: Hierbei wird ein Individuum dadurch motiviert, dass es erkannt hat, dass die jeweilige Aktivität für sich persönlich und dessen Entwicklung wichtig ist und einen gewissen Wert hat.

Integration: Wenn ein Individuum Motivation für eine Aktivität mitbringt, dessen Werte und Nutzen sich mit den eigenen persönlichen Werten und Bedürfnissen vereinen lassen, dann spricht man von einer sogenannten Integration. Die Merkmale der Aktivität lassen sich also in bereits vorhandene Wert- und Nutzenvorstellungen integrieren.

Seiteneffekte von extrinsischer Motivation

In der Lehre unterrichtet meist ein Lehrender mehrere Lernende. Aus diesem Grund ist es unwahrscheinlich, dass alle Lernenden das gleiche Maß an intrinsischer Motivation mitbringen. Oft kommen deshalb extrinsische Motivationstreiber zum Einsatz, um zusätzlich zu motivieren. Hier muss man allerdings Acht geben, denn extrinsische Motivation kann Seiteneffekte hervorrufen, durch welche intrinsische Motivation leidet (vgl. Deci, Koestner und Ryan 1999). Im Detail gilt dies für Motivation, welche durch externe Regulierungen generiert wird (vgl. Ryan und Deci 2000, S. 62).

Dies wird anhand eines Beispiels deutlich: Eine Person steckt viel Zeit in das Malen von qualitativ hochwertigen Bildern und ist intrinsisch motiviert, weil sie Spaß daran hat. Wenn man diese Person nun pro gemaltem Bild belohnen würde, dann würde diese Person je nach Belohnung dazu tendieren, die Quantität zu steigern, wodurch die Qualität leidet. Infolgedessen rutscht die Belohnung in den Vordergrund, wodurch der Spaß an der eigentlichen Aktivität leidet und somit die intrinsische Motivation negativ beeinflusst wird. Schlussendlich ist ein ähnliches Phänomen in der Lehre zu beobachten, wo häufig empfundene Ungerechtigkeiten in der Bewertung dazu führen, dass Lernende weniger intrinsisch motiviert sind.

2.1.3 Revised Bloom's Taxonomy

Vorab bestimmte Lernziele sind ein wichtiger Bestandteil der Lehre, um den Zweck einzelner Lehreinheiten zu begründen. Dabei ist es wichtig, die Fähigkeiten, welche in den Lehreinheiten vermittelt werden sollen, zu kategorisieren, damit jeweils passende Methoden und Vorgehen angewendet werden können. Somit wird außerdem sichergestellt, dass nicht nur Lehrende, sondern auch Lernende den Nutzen eben dieser Methoden und Vorgehen wahrnehmen. Eine solche Kategorisierung wurde 1956 unter dem Namen *Bloom's Taxonomy* bekannt. Auf dieser Basis entstanden viele Variationen, darunter die *Revised Bloom's Taxonomy* (vgl. Armstrong 2016). Diese fokussiert sich verstärkt auf kognitive Prozesse, sodass aus der Kategorie *Knowledge* beispielsweise *Remembering* wurde. Dieser Ansatz erlaubt außerdem basierend auf den verschiedenen kognitiven Prozessen eine gezieltere Evaluierung der Fähigkeiten von Lernenden. Die kognitiven Prozesse werden in sechs Kategorien unterteilt (vgl. Forehand 2010):

- *Remembering*: Erschließen, Merken und Abrufen von Wissen
- *Understanding*: Konstruieren einer Bedeutung auf Basis von Gesagtem, Geschriebenem und Visuellem
- *Applying*: Ausführen einer bestimmten Methodik auf neue Kontexte

- *Analyzing*: Bestandteile identifizieren und Zusammenhänge erkennen
- *Evaluating*: Beurteilen anhand bestimmter Kriterien oder Standards
- *Creating*: Verschiedene Elemente zu einem neuen funktionalen Gesamtwerk kombinieren

Im Gegensatz zu der ursprünglichen Version ist die Revised Bloom's Taxonomy nicht eindimensional. Es wird eine zweite Dimension eingeführt, welche Arten von Wissen kategorisiert und diesen verschiedene Tätigkeiten zuordnet, welche jeweils einer kognitiven Fähigkeit entsprechen. Wenn man beispielsweise die kognitive Fähigkeit *Creating* betrachtet, dann wird im Bereich des Faktenwissens kombiniert und im Bereich des Konzeptwissens geplant. Sowohl das Kombinieren als auch das Planen wird hierbei derselben kognitiven Fähigkeit zugeschrieben (vgl. Forehand 2010). Die Wissensdimension lässt sich in vier Kategorien unterteilen (vgl. Armstrong 2016):

- *Factual Knowledge*: Wissen über Terminologie, bestimmte Details oder Elemente
- *Conceptual Knowledge*: Wissen über Kategorien, Prinzipien, Theorien, Modelle und Strukturen
- *Procedural Knowledge*: Wissen über kontextabhängige Fähigkeiten, Vorgehen, Methoden, Techniken und Wissen über Kriterien, welche bestimmen, wann welche Methoden angewendet werden sollten
- *Metacognitive Knowledge*: Strategisches Wissen über sich selbst in Bezug auf eigene Denkweisen und das eigene Wissen

Im Rahmen dieser Arbeit sollen diese zwei Dimensionen dazu genutzt werden, um einzuordnen, welche Methoden und Werkzeuge im Bereich der modernen Programmierung geeignet sind, um spezifische kognitive Fähigkeiten auszubauen. Dies wird außerdem abhängig gemacht von einer bestimmten Wissensdimensionen.

2.1.4 Assessment

Das Ziel von Assessment ist es, den Lernprozess von Lernenden zu verbessern. Es geht weniger darum, die Fähigkeiten von Lernenden in Form von Noten zu beurteilen, sondern mehr darum, die Stärken und Schwächen von Lernenden herauszustellen und dort anzusetzen. Assessment sollte sich dabei stark an den Lernzielen orientieren und kontinuierliches Feedback liefern. Dieses Feedback soll zum einen Lehrenden aufzeigen, wie der derzeitige Stand der Lernenden ist und zum anderen individuellen Lernenden die Möglichkeit bieten, den eigenen Stand zu reflektieren (vgl. Hazzan, Ragonis und Lapidot 2020, S. 279f).

Formatives und Summatives Assessment

Formatives Assessment sollte während des ganzen Lehr- und Lernprozesses kontinuierlich durchgeführt werden. Dies führt dazu, dass Lernende schnell ihre Stärken und Schwächen identifizieren und auch dazu, dass Lehrende ihre Lehrmethoden dynamisch an den Lernfortschritt der Lernenden anpassen können.

Summatives Assessment wird zum Ende eines Lernprozesses durchgeführt. Diese Art des Assessments erlaubt es, den Lernerfolg von verschiedenen Lernenden miteinander zu vergleichen. Deswegen wird dieser Ansatz oft verfolgt, um unterschiedliche Leistungen zu benoten. Dies muss nicht zwingend der Fall sein. Vor allem wenn summatives Assessment mehrmals innerhalb eines Kurses durchgeführt wird, kann auch dieser Ansatz genutzt werden, um Lernprozesse zu steuern und nochmals Themen und Konzepte aufzugreifen, welche nicht gänzlich verstanden wurden (vgl. Hazzan, Ragonis und Lapidot 2020, S. 281).

Self- und Peer-Assessment

Self Assessment beschreibt das Beurteilen eigener Leistungen. Beim *Peer Assessment* hingegen werden Ergebnisse von anderen Lernenden beurteilt. Self- und Peer-Assessment können auch verknüpft werden, wobei dies die Fähigkeiten der Lernenden im Bereich der Beurteilung weiterentwickelt, weil aufgrund des zusätzlichen Peer-Feedbacks das eigene Feedback reflektiert werden kann (vgl. Hazzan, Ragonis und Lapidot 2020, S. 281). Schlussendlich können Lernende nicht nur aus dem Feedback lernen, welches sie erhalten, sondern sie werden zusätzlich mit dem Lernstoff konfrontiert, während sie anderen Feedback geben (vgl. Li 2011).

E-Assessment

Unter E-Assessment versteht man Assessments, bei denen elektronische Informations- und Kommunikationssysteme involviert sind (vgl. Ridgway, McCusker und Pead 2004, S. 41). Ein Teil des Assessments geschieht also automatisch, wodurch sich folgende Vorteile ergeben (vgl. Ridgway, McCusker und Pead 2004, S. 17ff):

- Ermöglicht höhere Kapazitäten von Lernenden aufgrund elektronischer Unterstützung, welche Lehrende entlastet
- Elektronische Verfahren für Assessment lassen sich gut in Bereiche eingliedern, wo in der Lehre ohnehin schon viele elektronische Werkzeuge Anwendung finden z.B. in der Programmierung
- E-Assessment kann auf Wunsch durchgeführt werden und ist unabhängig von den Zeitressourcen anderer

- Lernende bekommen individuelles Feedback
- Elektronisches Feedback ist sofort einsehbar
- Ist ein elektronisches Assessment Verfahren erst einmal implementiert, können Kosten eingespart werden oder Personalressourcen gezielter eingesetzt werden
- Lernende stehen in direktem Kontakt mit dem System und haben somit mehr Kontrolle. Ansätze wie Gamification können außerdem motivierend wirken
- Umfassend elektronisch verfügbare Ergebnisse von Assessments gestalten es Lehrenden leichter, sich einen Überblick über den derzeitigen Stand der Lernenden zu machen

Abgrenzung zu E-Learning

E-Assessment sollte allerdings nicht mit E-Learning verwechselt werden, denn E-Assessment soll Lernprozesse mithilfe von elektronischen Informations- und Kommunikationssystemen verbessern. E-Learning hingegen soll die Lernprozesse mithilfe von elektronischen Mitteln unterstützen und nicht die Lernprozesse an sich verbessern (vgl. Tavangarian u. a. 2004, S. 273).

2.1.5 Anforderungsliste

Aus den vorgestellten Konzepten der Didaktik können Anforderungen in Bezug auf die Gestaltung der praxisorientierten Lehre von Programmieransätzen und damit einhergehende Werkzeuge gewonnen werden. Diese Anforderungen werden zwar nicht strikt mithilfe einer Satzschablone formuliert, dennoch werden Muss-, Soll- und Kann-Vorschriften genutzt, um eine erste Priorisierung anzugeben (vgl. Bradner 1997). Es folgt die Anforderungsliste:

- *AD-01*: Lernende müssen gelehrte Konzepte und Ansätze moderner Programmierung praktisch anwenden
- *AD-02*: Lehreinheiten, welche neue Inhalte vermitteln, können nach dem Ansatz Active Learning gestaltet werden
- *AD-03*: Die intrinsische Motivation von Lernenden muss gefördert werden und sollte nicht durch den exzessiven Gebrauch von externen Regulierungen negativ beeinflusst werden
- *AD-04*: Von Lernenden zu bearbeitende praktische Aufgaben sollten mindestens den kognitiven Prozess *Applying* in den verschiedenen Wissensdimensionen voraussetzen und fördern

- *AD-05*: Lernende müssen im Rahmen der praxisorientierten Lehre stetig formatives Feedback erhalten
- *AD-06*: Lehrenden muss das formative und summative Feedback mit Bezug auf einzelne Lernende vorliegen
- *AD-07*: Sowohl Lernende als auch Lehrende sollten von dem Gebrauch geeigneter E-Assessment-Lösungen profitieren

2.2 Konzepte der Programmierung

Zur modernen Programmierung gehören auch das Verstehen und Anwenden von modernen Konzepten. Im Folgenden werden drei wichtige Konzepte kurz vorgestellt.

2.2.1 Domain Driven Design

Domain Driven Design stellt eine Herangehensweise zur Modellierung von komplexer Software dar, wobei sich dieser Ansatz stark an der zu modellierenden Fachlichkeit orientiert. Um dieses Ziel zu erreichen, vereint Domain Driven Design viele Konzepte und Design Ansätze. Zwei wesentliche Ansätze sind das *Strategic Design* und das *Tactical Design*.

Strategic Design
Wenn eine Domäne sehr komplexe Ausmaße annimmt, wird es schwierig diese im Ganzen zu verstehen. Mithilfe des Strategic Designs kann die Domäne unterteilt werden, sodass sich ein verständlicheres Gesamtbild ergibt (vgl. Evans 2004, S. 328). Die Domäne wird hierbei in sogenannte *Bounded Contexte* aufgeteilt, die jeweils einen bestimmten Funktionsumfang beschreiben und in ihrer Modellierung relativ unabhängig sind von anderen Bounded Contexten (vgl. Evans 2004, S. 335ff). Damit verschiedene Bounded Contexte aber ein Gesamtbild ergeben, müssen Arten von Schnittstellen zwischen diesen abgesprochen und definiert werden. Dies wird mittels einer *Context Map* abgebildet, welche es erlaubt, sich eine Übersicht über die gesamte Domäne zu verschaffen (vgl. Evans 2004, S. 344-353).

Tactical Design
Tactical Design geht innerhalb eines Bounded Contexts mehr ins Detail und beschreibt, wie die Fachlichkeit eines Bounded Contexts mithilfe von bestimmten *Building Blocks* abgebildet werden kann. Im Folgenden werden einige Building Blocks aufgezählt (vgl. Evans 2004, S. 89-158):

- Entities

- Value Objects
- Services
- Modules
- Aggregates
- Factories
- Repositories

Ubiquitous Language

Ubiquitous Language ist ein weiterer wichtiger Bestandteil des Domain Driven Designs. Ziel dabei ist es, ein einheitliches Verständnis für Begrifflichkeiten innerhalb der Domäne zu schaffen. Dies ist von äußerster Wichtigkeit, da es sonst oft zu Verständnisproblemen kommen kann. Ohne ein einheitliches Verständnis leiden außerdem die Kommunikation zwischen einzelnen Entwicklern und die Kommunikation zwischen Entwicklern und Experten innerhalb der Domäne (vgl. Evans 2004, S. 24-26).

2.2.2 Clean Code

In der Programmierung spielt *Clean Code* eine wichtige Rolle. Das Konzept betont die Relevanz und die Bedeutsamkeit von sauber geschriebenem Programmcode und sauberer Dokumentation. Wenn diesen Aspekten keine Priorität zugeschrieben wird, kann dies zu unsauberem Programmcode führen, welcher schlecht verständlich ist. Infolgedessen leidet unter diesem Umstand die Entwicklung von Programmen, welche unsauberen Programmcode enthalten. Handelt es sich um ein neues Projekt, so kann auch durch das Schreiben von unsauberem Programmcode die Entwicklung zu Anfang schnell voranschreiten. Je komplexer die Software allerdings wird, desto mehr beeinflusst unsauberer Programmcode das Tempo der Entwicklung in negativer Weise. Im schlimmsten Fall kann dies dazu führen, dass die Software nicht weiterentwickelt werden kann und neu geschrieben werden muss, um die Entwicklung weiterzuführen (vgl. Martin 2009, S. 3ff). Aus diesem Grund ist es von äußerster Wichtigkeit, dass Lernenden die Bedeutung und Relevanz von Clean Code möglichst früh vermittelt wird. Im gleichen Zug sollte den Lernenden natürlich auch beigebracht werden, mithilfe von welchen Konzepten oder Richtlinien auch sie lernen können, sauberen Programmcode zu schreiben.

Richtlinien und Prinzipien

Wenn jemand bloß die empfohlenen Richtlinien und Prinzipien kennt, welche zu Clean Code führen sollen, bedeutet dies aber nicht automatisch, dass diese Person auch sauberen

Programmcode schreibt. Man muss ein Gefühl für sauberen und auch schlechten Programmcode entwickeln und dafür muss man natürlich auch Erfahrungen mit schlechtem Programmcode gemacht haben. Besagte Richtlinien und Prinzipien unterstützen dabei die Entwicklung eines Verständnisses für sauberen und schlechten Programmcode (vgl. Martin 2009, S. 15).

Statische Code Analyse

Ein Teil der Richtlinien für Clean Code können mittels statischer Code Analyse und dem Anwenden von bestimmten Regeln automatisch überprüft werden. Einige Richtlinien sind dabei leichter prüfbar als andere. Beispielsweise ist mittels Software nicht automatisch prüfbar, ob Variablen aussagekräftige Namen gegeben wurden. Ob diese Variablen allerdings den gängigen Konventionen für Variablenbenennung entsprechen ist gut automatisch prüfbar. Ein Beispiel für eine solche Software ist *SonarQube*. SonarQube kann Entwickler automatisch auf Programmteile hinweisen, welche bestimmte konfigurierte Regeln verletzten. Wenn SonarQube in *Continuous Integration* Pipelines integriert ist, kann auf Wunsch außerdem verhindert werden, dass Programmcode akzeptiert wird, welcher nicht den gewünschten Kriterien entspricht (vgl. *SonarQube Documentation* o.D.).

2.2.3 Test Driven Development

Will man herausfinden, ob sich programmierte Funktionalität wie gewünscht verhält, kann die Software in Hinblick auf das gewünschte Verhalten getestet werden. Dies kann unter anderem manuell geschehen, wobei dies schnell sehr aufwendig sein kann, wenn man bestimmte Funktionalität mehrfach testen muss, um jene zu gewährleisten. Automatische Tests haben hierbei den Vorteil, dass diese einmal geschrieben werden, um anschließend beliebig oft ausgeführt zu werden. Ein automatischer Unit Test testet hierbei eine Funktionseinheit innerhalb einer Software.

Der Ansatz *Test Driven Development* fordert zusätzlich, dass Tests geschrieben werden, bevor die dazugehörige Funktionalität implementiert wird. Wenn man den Ansatz verfolgt, erst die Funktionalität zu implementieren und dann erst zu testen, führt dies leider oft dazu, dass nicht genügend oder sogar gar nicht automatisch getestet wird. Test Driven Development bewirkt allerdings nicht nur, dass mehr automatisch getestet wird, sondern hilft Entwicklern durch ein strukturiertes Vorgehen auch bei der Bewältigung von Problemstellungen mit hoher Komplexität. Umfangreiche Funktionalitäten werden mithilfe von Test Driven Development in kleinere Fragmente unterteilt, welche automatisch mithilfe vorab geschriebener Tests überprüft werden können (vgl. Beck 2002).

Vorgehen

Im Folgenden werden die einzelnen Schritte einer Iteration, welche im Rahmen des Test Driven Developments stattfinden, aufgezählt (vgl. Williams, Maximilien und Vouk 2003, S. 35):

1. Zuerst wird eine kleine Anzahl von automatischen Unit Tests geschrieben
2. Geschriebene Unit Tests werden ausgeführt und sollten nicht erfolgreich durchlaufen, weil zu überprüfende Funktionalität noch nicht implementiert wurde
3. Nun wird Programmcode geschrieben, welcher zu überprüfende Funktionalitäten implementiert
4. In einem letzten Schritt werden vorher geschriebene Unit Tests erneut ausgeführt und sollten nun erfolgreich durchlaufen, wenn die gewünschte Funktionalität entsprechend implementiert wurde. Gegebenenfalls werden alle vorhandenen Unit Tests ausgeführt, um sicherzustellen, dass durch den neuen Programmcode keine bestehenden Funktionalitäten beeinträchtigt werden (Regression Testing)

Mock Objects

Weil häufig Programmcode getestet wird, der noch nicht geschrieben ist, kann es dazu kommen, dass Tests nicht kompilieren. Hier können sogenannte *Mock Objects* Abhilfe schaffen, welche das Verhalten von Komponenten oder Ressourcen simulieren, welche noch nicht existieren (vgl. Mackinnon, Freeman und Craig 2000, S. 288). Außerdem sind besagte Komponenten oder Ressourcen häufig teuer in der Ausführung. Auch in diesem Fall eignen sich Mock Objects, um eigentliche Funktionalitäten der Komponenten oder Ressourcen zu simulieren (vgl. Beck 2002).

2.2.4 Anforderungsliste

Anforderungen an die Gestaltung der praxisorientierten Lehre von Programmieransätzen und damit einhergehende Werkzeuge werden nicht nur aus Konzepten der Didaktik, sondern auch aus modernen Konzepten der Programmierung gewonnen. Die Formulierung der Anforderungen orientiert sich dabei an den Leitlinien, welche bereits in Kapitel 2.1.5 vorgestellt wurden. Es folgt die Anforderungsliste:

- *AP-01*: Im Rahmen der praxisorientierten Lehre des Ansatzes Domain Driven Design sollte das Tactical Design, damit zusammenhängende Building Blocks und Ubiquitous Language thematisiert werden

- *AP-02*: Im Rahmen der praxisorientierten Lehre des Ansatzes Domain Driven Design kann das Strategic Design thematisiert werden
- *AP-03*: Lernenden muss die Bedeutung, Relevanz und Prinzipien von Clean Code vermittelt werden
- *AP-04*: Lernende müssen Feedback in Bezug darauf erhalten, inwiefern ihr Programmcode den gängigen Richtlinien für sauberen Programmcode entspricht
- *AP-05*: Im Rahmen von Programmieraufgaben müssen Lernende Programmcode gegen Unit Tests entwickeln
- *AP-06*: Lernenden sollte vermittelt werden, wie Unit Tests designt und implementiert werden
- *AP-07*: Lernenden kann vermittelt werden, welche Randbedingungen das Entwickeln von Software mithilfe des Ansatzes Test Driven Development mit sich bringt und wie dabei vorgegangen wird

3 Experteninterviews

Auf Basis vorgestellter didaktischer Konzepte und moderner Programmieransätze liegen bereits einige Anforderungen an die praxisorientierten Lehre im Bereich der modernen Programmierung vor. Das Ganze sollte allerdings nochmal aus verschiedenen Blickwinkeln betrachtet werden, um zusätzliche Anforderungen zu erheben. Dafür wäre es interessant, das Wissen und die Erfahrungen von anderen Lehrenden beziehungsweise Experten auf diesem Gebiet heranzuziehen. Um dieses Wissen und diese Erfahrungen zu rekonstruieren und zu sammeln eignen sich Experteninterviews (vgl. Pfadenhauer 2002, S. 113). In diesem Kapitel wird zuerst das Vorgehen bei den Experteninterviews beschrieben, um anschließend auf die Ergebnisse eingehen und neue Anforderungen ableiten zu können.

3.1 Vorgehen

Die Experteninterviews wurden im Rahmen dieser Arbeit nach einem bestimmten Vorgehen durchgeführt, um mithilfe eines systematischen Ablaufs die bestmöglichen Ergebnisse erzielen zu können. Im Folgenden werden die verschiedenen Phasen erläutert, welche sich als essentiell für die Durchführung und Verarbeitung von Experteninterviews herausgestellt haben.

3.1.1 Expertenakquise

Bevor Experteninterviews durchgeführt werden können, müssen auf Basis bestimmter Kriterien Experten akquiriert werden. Dabei ist keine repräsentative Stichprobe an Experten nötig, denn das Ziel der Experteninterviews ist keine Generalisierung, sondern eine qualitative Analyse der Ergebnisse. Auf dieser Grundlage und weil Experteninterviews nicht zentraler Gegenstand dieser Arbeit sind, wurden die Experteninterviews in relativ kleinem Rahmen durchgeführt. Infolgedessen wurden insgesamt vier Experten befragt. Bei der Auswahl von Experten muss des Öfteren die Frage gestellt werden, welche relevanten Informationen von welchem Experten in Erfahrung gebracht werden können (vgl. Kaiser 2014, S. 71f). Deshalb wurden Experten ausgewählt, welche Einblicke aus verschiedenen Perspektiven in Bezug auf praxisorientierte Kurse mit Fokus auf Programmierung ermöglichen. Somit handelt es sich bei den Experten um Lehrende, welche Kurse gestalten, die einen jeweils anderen Wissensstand voraussetzen. Manche dieser Kurse sind Programmierkurse für Anfänger und andere wiederum setzen gewisses Basiswissen im Bereich der Programmierung voraus.

3.1.2 Erstellung des Interviewleitfadens

Ein wichtiger Bestandteil von Experteninterviews bildet der Interviewleitfaden. Dieser gilt als zentrales strukturierendes Element der Experteninterviews und bildet den Forschungsgegenstand in Form von Interviewfragen ab. Der Prozess des Übersetzens von Forschungsfragen zu Interviewfragen kann als *Operationalisierung* bezeichnet werden. Hierbei wird allerdings weiter zwischen *konzeptioneller Operationalisierung* und *instrumenteller Operationalisierung* unterschieden.

Konzeptionelle Operationalisierung
Die konzeptionelle Operationalisierung hat die Aufgabe, das Forschungsproblem zu konkretisieren. Hierfür werden zuerst *Analysedimensionen* aus den Forschungsfragen gewonnen, welche bestimmten, was beobachtet wird (siehe Anhang 2.1.1). Danach werden *Fragenkomplexe* identifiziert, welche bestimmen, nach welchen Kriterien eine Analysedimension beobachtet werden soll (siehe Anhang 2.1.2).

Instrumentelle Operationalisierung
Im Rahmen der instrumentellen Operationalisierung werden aus den Fragenkomplexen Interviewfragen gebildet. Diese haben nicht nur die Aufgabe, den Forschungsgegenstand für die Experten nachvollziehbar zu machen, sondern können je nach Formulierung und Fragentyp dazu beitragen, ein gewünschtes Ziel zu erreichen (vgl. Kaiser 2014, S. 52-63). Dieser Prozess und die daraus resultierenden Interviewfragen sind dem Anhang zu entnehmen (siehe Anhang 2.1.3).

3.1.3 Durchführung und Auswertung der Experteninterviews

Es gibt viele Wege, ein Interview zu führen, allerdings mussten aufgrund der zur Zeit der Verfassung dieser Arbeit aktuellen Corona-Lage sämtliche Interviews online über Zoom geführt werden. Die vier durchgeführten Interviews haben im Schnitt jeweils etwas weniger als eine Stunde gedauert. Am Anfang eines jeden Interviews wurde kurz der Forschungsgegenstand erläutert und bestimmte Formalitäten geklärt. Dabei wurden die Interviewten außerdem darüber in Kenntnis gesetzt, dass das Gesprochene aufgezeichnet und später transkribiert wird. Daraufhin folgten selbstverständlich die Interviewfragen, welche das eigentliche Kernstück der Interviews bildeten. Die Interviews wurden innerhalb einer Zeitspanne von etwa sechs Wochen geführt.

Transkription der Experteninterviews
Um die Ergebnisse von Interviews analysieren und interpretieren zu können, müssen die Interviews vorher transkribiert werden. Dabei wurde eine vereinfachte Transkription durch-

geführt, um Formulierungen zu vereinfachen und somit die Lesbarkeit zu fördern. Infolgedessen wurde das Gesagte weitestgehend wörtlich übernommen. Zugunsten der Lesbarkeit wurden außerdem Satzzeichen passend gesetzt. Die fertigen Transkripte sind im Anhang zu finden (siehe Anhang 2.2).

Kodierung der Interviewtranskripte
Bevor Interviewtranskripte kodiert werden, sollten in einem ersten Schritt Transkripte von Interviews herausgefiltert werden, welche als misslungen eingestuft werden. Da dies bei keinem der vier Interviews der Fall war, werden alle vier Interviews kodiert und ausgewertet. Im Zuge der Kodierung werden einzelne Textpassagen jeweils einer Kategorie zugeordnet. Somit kann erkannt werden, wann Stellungnahmen desselben oder verschiedener Experten thematisch zusammengehören, sodass eine systematische Analyse der Interviews möglich ist. Weil die Interviews relativ nah an dem Interviewleitfaden geführt wurden, genügen die Fragenkomplexe als Kategorien für die Kodierung (vgl. Kaiser 2014, S. 101-105). Eine kleine Anpassung muss bei der Kategorie *Moderne Konzepte und Herangehensweisen* vorgenommen werden. Diese wird weiter in *Domain Driven Design*, *Clean Code* und *Test Driven Development* unterteilt, weil dieser Kategorie ansonsten zu viele Textpassagen zugeordnet werden würden.

Interpretation der Ergebnisse
Sobald die Interviewtranskripte in kodierter Form vorliegen, können die Ergebnisse interpretiert werden. Die Erkenntnisse aus den Experteninterviews müssen dabei in Bezug zu im Rahmen dieser Arbeit relevanten Theorien und Konzepten gesetzt und anschließend analysiert werden (vgl. Kaiser 2014, S. 115). Dies wird in den nächsten Kapiteln geschehen, welche sich unter anderem auf die Ergebnisse der Interviews beziehen werden, um Lösungsvorschläge vorzustellen und Empfehlungen auszusprechen. Durch die vorgenommene Kodierung wird dabei das Finden von thematischen Zusammenhängen vereinfacht.

3.2 Anforderungsbereiche

Im Folgenden wird auf einen Teil der Erkenntnisse aus den Experteninterviews eingegangen, aus welchen sich Anforderungen in Bezug auf die praxisorientierte Lehre von modernen Programmieransätzen und damit einhergehende Werkzeuge ergeben.

3.2.1 Moderne Programmierung

Im Rahmen dieser Arbeit werden Ansätze thematisiert, welche sich mit der praxisorientierten Lehre von moderner Programmierung beschäftigen. Deshalb sollte festgehalten

werden, was die wichtigsten Merkmale moderner Programmierung sind, um jeweilige Lernziele daraus ableiten zu können.

In erster Linie bedeutet moderne Programmierung, dass aktuelle Technologien verwendet werden. Darüber hinaus sollten Alternativen in Betracht gezogen werden und vorhandene *Frameworks* oder *Libraries* in Hinblick auf den Nutzen für ein jeweiliges Projekt untersucht werden, um diese anschließend gegebenenfalls einzubinden. So kann Zeit eingespart werden und schnell eine vorzeigbare Lösung generiert werden (vgl. Interviewter 1 2020, Interview, siehe Anhang 2.2.1, Z. 53-69). Moderne Programmierung definiert sich allerdings nicht nur über aktuelle Technologien, sondern auch über bestimmte Herangehensweisen und Konzepte. Bente (2020, Interview, siehe Anhang 2.2.3, Z. 26-31) führt außerdem an, dass nachhaltiger Code, welcher unter Berücksichtigung der Clean Code Richtlinien geschrieben wird, ein grundlegendes Merkmal modernen Vorgehens ist.

Weitere typische Herangehensweisen umfassen das Domain Driven Design und das Test Driven Development. Diese Herangehensweisen beschreiben ein iteratives Vorgehen, welches eine Entwicklung nah an der gewünschten Fachlichkeit ermöglicht (vgl. Bente 2020, Interview, siehe Anhang 2.2.3, Z. 35-53).

Generell sollten im Rahmen von moderner Programmierung bewährte Entwurfsmuster und Prinzipien berücksichtigt werden (vgl. Kohls 2020, Interview, siehe Anhang 2.2.2, Z. 49-79). Reitano (2021, Interview, siehe Anhang 2.2.4, Z. 37-47) merkt zudem an, dass moderne Programmierung immer mehr ein sogenanntes *People Problem* ist. Projekte werden immer komplexer und erfordern somit mehr Personalressourcen. Infolgedessen ist nicht nur die technische Umsetzung von Bedeutung, sondern Teamarbeit gewinnt immer mehr an Relevanz.

3.2.2 Die Wahl der Programmiersprache

Um bei einem praxisorientierten Kurs den Vorbereitungsaufwand im Rahmen zu halten und Lernende nicht zu überfordern, wird in den meisten Kursen eine einzige Programmiersprache ausgewählt und somit Lehrinhalte anhand dieser Programmiersprache vermittelt. Je nach Lernzielen müssen dann Überlegungen angestellt werden, welche Programmiersprache für den Kurs am besten geeignet ist. Laut Kohls (2020, Interview, siehe Anhang 2.2.2, Z. 105-128) spielen drei Faktoren eine wichtige Rolle, wenn es um die Auswahl einer Programmiersprache in der Lehre geht. Zum einen muss die Programmiersprache industrienah beziehungsweise weit verbreitet und relevant sein. Bei einer abgeschlossenen Hochschulausbildung sind Kenntnisse in Programmiersprachen hilfreich, welche in

der Industrie Anwendung finden und somit benötigt werden. Zweitens sollte die Programmiersprache moderne Konzepte vereinen, um diese lehren zu können. Drittens sollte die Programmiersprache didaktisch geeignet sein.

Der dritte Faktor wird zudem durch eine Aussage von Reitano (2021, Interview, siehe Anhang 2.2.4, Z. 63-79) verdeutlicht. Diese besagt, dass es wichtig ist, dass Konzepte, welche gelehrt werden sollen, deutlich anhand der Programmiersprache identifizierbar und erklärbar sein müssen. Als Beispiel führt Reitano an, dass Objektorientierung nicht anhand der Programmiersprache *JavaScript* erklärt werden sollte, weil JavaScript zu viele Freiheiten einräumt, welche es erlauben, Objektorientierung an manchen Stellen falsch umzusetzen.

Natürlich sollte Lernenden stets der Standpunkt vermittelt werden, dass man bei der Wahl der Programmiersprache offen und flexibel sein sollte und je nach Problemstellung eine Programmiersprache besser geeignet sein kann als eine andere (vgl. Bente 2020, Interview, siehe Anhang 2.2.3, Z. 60-78).

3.2.3 Essenzielle Werkzeuge zur Aufgabenbearbeitung

Zu den Fähigkeiten eines Programmierers gehört nicht nur das Programmieren selbst, sondern auch das Nutzen von Werkzeugen, welche das Erstellen von Software unterstützen. Deshalb sollte es ein Lernziel sein, Lernende möglichst früh mit dementsprechenden Werkzeugen zu konfrontieren. Das wohl wichtigste Werkzeug bildet die Entwicklungsumgebung mitsamt integrierter Tools wie beispielsweise den *Debugger*. Neben dieser darf auch die Versionsverwaltung über zum Beispiel *Git* nicht fehlen. Da Teamarbeit in der Programmierung eine immer wichtigere Rolle einnimmt, sollten auch Werkzeuge zum Projektmanagement thematisiert werden (vgl. Kohls 2020, Interview, siehe Anhang 2.2.2, Z. 5-25).

Es empfiehlt sich, Programmieranfänger Versionsverwaltung über die Entwicklungsumgebung zu betreiben, um diese anfangs nicht zu überfordern. Später sollten Erfahrungen mit der Bedienung von Git über die Konsole gesammelt werden, um die Konzepte und Funktionsweise dahinter besser nachvollziehen zu können. Somit werden Vorgehen transparent gemacht, welche zuvor hinter Funktionen der Entwicklungsumgebung verborgen waren (vgl. Kohls 2020, Interview, siehe Anhang 2.2.2, Z. 33-44).

Bente (2020, Interview, siehe Anhang 2.2.3, Z. 4-16) erwähnt außerdem Werkzeuge zur Realisierung von Continuous Integration Pipelines, welche durch Automatismen wiederkehrende Tätigkeiten ausführen, sodass eine effizientere Softwareentwicklung ermöglicht

wird. Lernende sollten zusätzlich mit Werkzeugen zur Dokumentationsgenerierung wie beispielsweise über *Gitlab-Pages* vertraut gemacht werden.

Schlussendlich wird in der Hochschulausbildung häufig vernachlässigt, Lernenden beizubringen, ein Produkt von Anfang bis Ende zu entwickeln und dieses anschließend auszuliefern. Dieser Prozess und dazugehörige Werkzeuge sollten ebenfalls Teil der Hochschulausbildung sein (vgl. Interviewter 1 2020, Interview, siehe Anhang 2.2.1, Z. 22-29).

Im Folgenden werden die hier aufgezählten Werkzeuge kurz zusammengefasst:

- Entwicklungsumgebung mitsamt integrierter Werkzeuge
- Versionsverwaltung über z.B. Git
- Werkzeuge zum Projektmanagement
- Werkzeuge zur Realisierung von CI Pipelines
- Werkzeuge zur Dokumentationsgenerierung

3.2.4 Gruppenarbeit bei der Aufgabenbearbeitung

Gruppenarbeit bringt einige Vorteile mit sich, auf welche an dieser Stelle allerdings nicht weiter eingegangen wird. Ein Problem bei Gruppenarbeit in Praktika ist jedoch, dass sich Lernende oft auf die Ergebnisse anderer verlassen. Kohls (2020, Interview, siehe Anhang 2.2.2, Z. 248-262) bevorzugt deshalb Gruppenarbeit mit einzelnen Abgaben und zusätzlichen mündlichen Abfragen. Auf diese Weise soll sichergestellt werden, dass sich jeder Lernende mit den Aufgaben beschäftigt hat.

Allerdings ist laut Bente (2020, Interview, siehe Anhang 2.2.3, Z. 204-231) Gesagtes in mündlichen Prüfungen oft schwer quantifizierbar. Somit sind häufig die formalen Kriterien für ein Bestehen erfüllt, dennoch wird deutlich, dass der Lernende nicht die nötige Ahnung von der Materie hat. Deshalb bietet es sich an, die Aufgaben in einem Maß zu individualisieren, welches zwar Diskussionen und Zusammenarbeit zwischen verschiedenen Lernenden ermöglicht, aber dennoch das Kopieren von Lösungen anderer erschwert. Somit muss sich jeder mit der Aufgabe beschäftigen und auch wenn sich Lernende an den Lösungen anderer bedienen, müssen diese Lösungen zumindest nachvollzogen und abgewandelt werden.

Auch wenn Gruppenarbeit in vielen Kursen eine Bereicherung darstellt, so sollte bei der Lehre von Programmiergrundlagen weniger auf Gruppenarbeit gesetzt werden. Diese Grundlagen muss jeder beherrschen und zugleich ein Verständnis für Programmcode entwickeln (vgl. Reitano 2021, Interview, siehe Anhang 2.2.4, Z. 138-150). In diesem Fall eignet sich der gelegentliche Einsatz von Techniken wie beispielsweise *Pair-Programming*, um das eigene Verständnis von Programmcode mit dem Verständnis anderer abzugleichen. Laut Kohls (2020, Interview, siehe Anhang 2.2.2, Z. 267-277) ist es allerdings schwierig, Lernende dazu zu bringen, sich an die Prinzipien und Vorgaben von Pair-Programming zu halten.

3.2.5 Anforderungsliste

In den vorherigen Kapiteln werden zusätzliche Anforderungsbereiche aufgegriffen, aus welchen sich neue Anforderungen ergeben. Diese erweitern die bestehende Anforderungsbasis, welche in den Kapiteln 2.1.5 und 2.2.4 beschrieben wird. Die Formulierung der Anforderungen orientiert sich dabei an den Leitlinien, welche bereits in Kapitel 2.1.5 vorgestellt werden. Es folgt die Anforderungsliste:

- *AE-01*: Lernende müssen mit modernen Technologien, Frameworks und Libraries konfrontiert werden
- *AE-02*: Lernende müssen mit bewährten Entwurfsmustern und Prinzipien aus der modernen Programmierung vertraut gemacht werden und diese praktisch anwenden
- *AE-03*: Lernende müssen im Rahmen von praxisorientierten Programmierkursen ihre Fähigkeiten im Bereich der Teamarbeit ausbauen
- *AE-04*: In der Lehre thematisierte Programmiersprachen müssen moderne Konzepte der Programmierung vereinen, in der Industrie relevant und didaktisch geeignet sein
- *AE-05*: Lernende müssen Werkzeuge wie die Entwicklungsumgebung und Werkzeuge zur Versionsverwaltung nutzen
- *AE-06*: Lernende sollten Werkzeuge zum Projektmanagement und Werkzeuge zur Realisierung von CI Pipelines und zur Dokumentationsgenerierung nutzen
- *AE-07*: Lernende sollten mit der Funktionsweise moderner Werkzeuge zur Programmierung und damit verbundenen Konzepten vertraut gemacht werden
- *AE-08*: Auch wenn Lernende in Gruppen zusammenarbeiten, sollten die Abgaben einzeln erfolgen

- *AE-09*: Aufgaben sollten in einem Maß individualisiert werden, welches zwar Diskussionen und Zusammenarbeit zwischen verschiedenen Lernenden ermöglicht, aber dennoch das Kopieren von Lösungen anderer erschwert
- *AE-10*: Bei der Lehre von Programmiergrundlagen sollte weniger auf Gruppenarbeit gesetzt werden
- *AE-11*: Lernende können mit dem Ansatz Pair-Programming konfrontiert werden und ihr Verständnis von Programmcode mit dem Verständnis anderer abgleichen

4 Existierende Lösungen

In diesem Kapitel wird zum einen auf bereits existierende Ansätze und Werkzeuge eingegangen, welche sich mit dem Automatisieren und Individualisieren von praxisorientierten Übungen beschäftigen. Zum anderen werden Werkzeuge zur Kommunikation thematisiert, welche digitale Anteile von Praktika unterstützen sollen. Die Auswahl der Werkzeuge erfolgt dabei auf Basis der erhobenen Anforderungen aus den Kapiteln 2.1.5, 2.2.4 und 3.2.5.

4.1 Automatisierung und Individualisierung

Wenn man in Kapitel 2.1 beschriebene didaktische Konzepte in der praxisorientierten Lehre berücksichtigen möchte, ist an einigen Stellen Unterstützung durch diverse Werkzeuge hilfreich. Werkzeuge zur Automatisierung begünstigen E-Assessment und erlauben das Integrieren von komplexen mittels Tests automatisch überprüfbaren Aufgaben in den Lehrplan. Dahingehen erschweren es Werkzeuge zur Aufgabenindividualisierung Lernenden, die Lösungen anderer zu kopieren. Auf bestimmte Konzepte in der Programmierung wie beispielsweise Clean Code (siehe Kapitel 2.2.2) kann mittels gezielter Tests vermehrt aufmerksam gemacht werden. Je nach Gestaltung der Aufgaben und damit verbundenen Tests kann hierbei eine hohe Taxonomiestufe erreicht werden.

4.1.1 Individualisierung in der Mathematik

Im Bereich der Mathematik existieren bereits viele Lösungen für automatisierte Systeme, welche auch Aufgaben individualisieren. Im Folgenden werden zwei Lösungen vorgestellt, um Prinzipien aufzuzeigen, welche eventuell für den Bereich der Informatik adaptiert werden können.

Parametrisierte Aufgaben

Um mathematische Aufgaben zu individualisieren, können diese parametrisiert werden. Dies bedeutet, dass die Lösung einer Aufgabe nicht nur durch konkrete Zahlen bestimmt wird. Zusätzlich werden Variablen genutzt, welchen während der Aufgabenerstellung zufällig jeweils ein bestimmter Wert innerhalb eines festgelegten Wertebereichs zugeordnet wird. Werte, welche man einer Variable zuordnet, werden entweder als Konstante oder als evaluierte Konstante definiert. Konstanten sind dabei konkrete Zahlen wie beispielsweise *1,2,7,11*. Evaluierte Konstanten basieren auf den Werten von vorab definierten Konstanten und werden durch einen Term wie beispielsweise $A+B$ definiert. Variablen und dessen Wertebereiche werden in Form einer *XML*-Datei konfiguriert und auf dessen Basis zufällig

zugewiesen (vgl. Goguadze 2010, S. 77-83). Die Art und Weise der Variablenkonfiguration wird in nachfolgender Darstellung verdeutlicht:

```
<exercise_generator name="Randomizer">
  <parameter name="constant">
    A,1,2,3,4,5,6,7,8,9,10
  </parameter>
  <parameter name="constant">
    B,1,2,3,4,5,6,7,8,9,10
  </parameter>
  <parameter name="constant">
    A_plus_B, A+B
  </parameter>
  <parameter name="evaluated_constant">
    A_plus_B_evaluated, A+B
  </parameter>
</exercise_generator>
```

Abbildung 1: Konfiguration von Variablen für parametrisierte Aufgaben (Goguadze 2010, S. 78)

Generieren der Aufgabenstellung

Eine weitere Herausforderung bei individualisierten Aufgaben ergibt sich durch die Aufgabenstellung. Diese wird bestenfalls passend der individualisierten Aufgabe ebenfalls individualisiert. Die Toolbox *MATEX* nimmt sich diesem Problem an und verknüpft das Software-Paket *MATLAB* mit *LATEX*-Funktionalitäten (vgl. Helfrich-Schkarbanenko u. a. 2018). Die Variablen, welche die verschiedenen Variationen bestimmen, werden hier mittels der MATLAB-Funktion *randi* erzeugt (vgl. Helfrich-Schkarbanenko u. a. 2018, 58ff). Um auch die Aufgabenstellung anhand dieser Variablen zu generieren, stellt MATEX einige Hilfsfunktionen zur Verfügung, welche während der Aufgabenindividualisierung LATEX-Dateien mit Inhalt füllen (vgl. Helfrich-Schkarbanenko u. a. 2018, S. 46–49).

4.1.2 Ilias

Ilias ist eine Open-Source E-Learning Software, welche unter anderem Funktionalitäten zur Durchführung von E-Assessments (siehe Kapitel 2.1.4) beinhaltet. Somit wird sowohl formatives und summatives Feedback in Form von Lernfortschrittskontrollen verfügbar gemacht, als auch Self- und Peer-Assessment mittels integrierter Funktionalitäten ermöglicht. Mithilfe der Plattform können innerhalb einer Instanz verschiedene Kurse abgebildet werden. Für jeden Kurs kann Lernmaterial zur Verfügung gestellt werden. Bereitgestellte

Übungen lassen sich dem E-Learning zuordnen. Durchzuführende Tests zeigen den eigenen Lernfortschritt und aus Sicht von Lehrenden den Lernfortschritt vieler Lernender auf. Im Gegensatz zu Übungen sind Tests auf Basis bestimmter Fragentypen außerdem so konzipiert, dass Tests automatisch ausgewertet werden können. Im Folgendem werden einige mögliche Fragentypen aufgezählt (vgl. *Dokumentation für Autoren* o.D.):

- Single-Choice-Fragen
- Multiple-Choice-Fragen
- Lückentexte
- Numerische Fragen
- Anordnungsfragen
- Zuordnungsfragen
- Freitext-Fragen
- Datei-Upload-Fragen

Individualisierung mittels Fragenpools
Ilias bietet die Möglichkeit, Tests mittels Fragenpools zu individualisieren. Fragen werden von Lehrenden erstellt und im Fragenpool abgelegt. Wenn ein Lernender nun einen Test startet, welcher auf einem Fragenpool basiert, dann werden aus dem Fragenpool zufällige Fragen ausgewählt (vgl. *Dokumentation für Autoren* o.D.). Um während des Tests Lösungen von anderen Lernenden kopieren zu können, müsste man die Lösungen eines anderen Lernenden, welcher die gleichen Fragen gewürfelt hat, erhalten. Je größer der Fragenpool ist, desto schwieriger wird das Auftreiben von Lösungen für Fragen, welche einem einzelnen Lernenden zugewiesen wurden.

Daraus folgt jedoch ein großes Problem von Item Pools, denn es sind meist viele Test-Items innerhalb eines Pools nötig, um von den Vorteilen dieser Art von Individualisierung profitieren zu können. Auch wenn Item Pools über eine längere Zeit hinweg stetig aufgebaut werden können, sollte der dafür benötigte Zeitaufwand nicht unterschätzt werden. Ein anderes Problem stellen abweichende Schwierigkeitsgrade dar, welche sich aus zufällig gewählten Test-Items ergeben. Zwar gibt es Ansätze zur Sicherstellung ähnlicher Schwierigkeitsgrade, jedoch darf nicht vergessen werden, dass diese Ansätze auf einem korrekten Abschätzen von Schwierigkeiten einzelner Test-Items basieren (vgl. Ras und Joosten-ten Brinke 2015, S. 47-51).

Programmieraufgaben mittels Ilias

Das Abbilden von praxisorientierten Aufgaben im Bereich der Programmierung innerhalb der Lernplattform Ilias gestaltet sich schwierig. Zwar könnten Programmieraufgaben als Ilias-Übungen von integrierten Self- und Peer-Feedback Verfahren profitieren, jedoch bieten die vorgestellten Fragentypen, welche in Tests zum Einsatz kommen, keine Möglichkeit, diese Programmieraufgaben automatisch zu kontrollieren oder Feedback zu generieren. Natürlich kann mittels dieser Fragentypen bestimmtes Wissen über Konzepte oder Methoden aus der Programmierung abgefragt werden, wenn es allerdings um das Programmieren als Disziplin als solche geht, so kann dies nicht mittels jener Fragentypen abgebildet werden. Dies kann gezielter mithilfe der Revised Bloom's Taxonomy (siehe Kapitel 2.1.3) eingeordnet werden. Je weiter abzufragendes Wissen vom *Factual Knowledge* bis hin zum *Procedural Knowledge* einzuordnen ist, desto schwieriger wird es komplexere kognitive Prozesse wie beispielsweise *Creating* durch Fragentypen aus Ilias-Tests abzubilden. Um Programmieren zu lernen, ist es aber wichtig, diese Ebene der kognitiven Prozesse zu erreichen. Natürlich können Ilias-Tests unterstützend genutzt werden, besonders gut eignen sich Ilias-Fragen dabei in den Bereichen *Remembering* und *Understanding*.

4.1.3 Gitlab

Gitlab ist primär eine Plattform zur Verwaltung von Programmcode, welche selbst gehostet werden kann (vgl. *GitLab Docs* o.D.). Somit kann Gitlab auch als zentrale Plattform für Programmieraufgaben genutzt werden. Die Rechteverwaltung erlaubt die Vergabe von Zugriffsrechten auf Basis von Gruppen oder Nutzern (vgl. Van Baarsen 2014). Für einen Programmierkurs kann daher eine Gruppe angelegt werden, in welcher sich Aufgaben für Lernende befinden. Diese können die Aufgaben dann herunterladen und bearbeiten. Wenn Aufgaben individuell abgegeben werden sollen, empfiehlt sich das Anlegen von separaten Projekten für einzelne Lernende. Dies wird ermöglicht durch die Zugriffsverwaltung auf Nutzerbasis, sodass jeder Lernende nur sein eigenes Projekt sehen kann. Als Lehrender hat man aufgrund von Admin-Rechten Zugriff und Einsicht auf sämtliche Projekte von Lernenden. Da Gitlab auf Git basiert, werden Kenntnisse über den Git Workflow und die Git Befehle benötigt, um auf Gitlab gestellte Aufgaben zu bearbeiten. Lernende werden insofern dazu verleitet, sich mit Git zu beschäftigen. Ob dies von Vorteil ist, hängt allerdings immer von den jeweiligen Lernzielen ab. Schlussendlich ist noch anzumerken, dass Gitlab, auch wenn individuelle Projekte für Lernende unterstützt werden, über keinerlei Automatismus verfügt, welcher es erlaubt, für mehrere Lernende automatisch eigene Projekte anzulegen.

Automatische Überprüfung von Aufgaben

Mittels *Gitlab CI* hat man die Möglichkeit, eine Continuous Integration Lösung in Gitlab zu integrieren. Somit können relativ einfach CI-Pipelines für einzelne Projekte definiert werden. Diese ermöglichen zugleich das Ausführen von automatischen Unit Tests, welche angestoßen werden, sobald ein *Commit* registriert wird (vgl. Van Baarsen 2014). Mittels Unit-Tests können von Lernenden hochgeladene Lösungen in Hinblick auf verschiedene Kriterien automatisch überprüft werden. Infolgedessen erhalten Lernende und auch Lehrende schnelles Feedback in Bezug auf die Richtigkeit hochgeladener Lösungen. Nicht erfolgreiche Unit-Tests können außerdem die Schwächen von Lernenden aufzeigen und diese dazu bewegen, an diesen zu arbeiten. Insgesamt hat diese Art des Feedbacks sowohl summativen als auch formativen Charakter (siehe Kapitel 2.1.4).

Diese Art der automatischen Überprüfung lässt sich außerdem von den Tests innerhalb der Lernplattform Ilias (siehe Kapitel 4.1.2) abgrenzen. Im Vergleich zu den Fragentypen von Ilias bieten Unit-Tests viel mehr Möglichkeiten, Programmcode in Hinblick auf verschiedene Kriterien zu testen. Zwar sind bei Aufgaben, welche mittels Unit Tests getestet werden sollen, bestimmte Vorgaben nötig, jedoch begünstigt alles, was jenseits dieser Vorgaben liegt, die Freiheit von Lernenden bei der Aufgabenbearbeitung. Somit ist es je nach Aufgabenstellung relativ einfach höhere kognitive Prozesse wie beispielsweise den Prozess *Creating* in sämtlichen Wissensdimensionen anzustoßen (siehe Kapitel 2.1.3).

4.1.4 Praktomat

Auch wenn Ilias (siehe Kapitel 4.1.2) zur Überprüfung von Programmieraufgaben nicht genügt und Gitlab (siehe Kapitel 4.1.3) dahingehend allein zu wenig Automatisierung bietet, gibt es einige andere Werkzeuge, welche sich diesem Problem widmen. Eines dieser Werkzeuge nennt sich *Praktomat*. Dieser ist dem E-Assessment (siehe Kapitel 2.1.4) zuzuordnen, denn er vereint Feedback von automatischen Tests mit Peer-Feedback von anderen Lernenden und Feedback von Lehrenden. In Hinblick auf die Revised Bloom's Taxonomy (siehe Kapitel 2.1.3) sind Programmieraufgaben, welche mithilfe des Praktomats überprüft werden, gleich einzuordnen wie Programmieraufgaben über Gitlab. Zusätzlich werden durch den Praktomat Funktionalitäten zur automatischen Erstellung von Aufgabenvarianten bereitgestellt (vgl. Krinke, Störzer und Zeller 2002).

Automatische Überprüfung von Aufgaben

Wie bereits erwähnt überprüft der Praktomat Aufgaben mittels automatischen Tests. Dabei wird zwischen verpflichtenden, optionalen und versteckten Unit-Tests unterschieden. Verpflichtende Tests sind erforderlich, um eine Aufgabe zur Bewertung durch einen

Lehrenden freizugeben. Optionale und versteckte Tests geben den Lernenden die Möglichkeit, ihre Lösung stetig zu verbessern und sollen dazu motivieren, ausführlicher zu Testen. Schlagen Tests fehl, kann ein Lernender seine Lösung beliebig oft verbessern, um die Testfälle zu bestehen, auch wenn sein Programm bereits zur Bewertung freigegeben wurde. Neben Funktionstests kann auch die Einhaltung von Clean Code Richtlinien (siehe Kapitel 2.2.2) mittels Tests überprüft werden. Auch hier gilt die Unterteilung zwischen verpflichtenden, optionalen und versteckten Tests. So können Tests bezüglich dieser Richtlinien anfangs optionaler Natur sein und später verpflichtend gemacht werden (vgl. Krinke, Störzer und Zeller 2002).

Dieser ganze Testautomatismus wird mittels sogenannter *Checker* ermöglicht. Jeder Checker nimmt dabei eine eigene Rolle in diesem Automatismus ein. Bestimmte Checker kompilieren abgegebenen Code, andere prüfen Funktionalität oder die Einhaltung von Clean Code Richtlinien. Aufgrund dieses Konzepts lässt sich der Praktomat relativ einfach auf verschiedene Programmiersprachen oder Aufgabenkontexte erweitern (vgl. Breitner, Hecker und Snelting 2017).

Individualisierung von Aufgaben
Mithilfe eines Makromechanismus können Aufgaben individualisiert werden, um einfaches Kopieren von Lösungen zu erschweren. Dafür werden die Aufgabenstellung und die Tests mit M4 Makroanweisungen versehen, sodass je nach individualisierter Variante eine andere Aufgabenstellung generiert wird und dazugehörige Tests ausgeführt werden (vgl. Krinke, Störzer und Zeller 2002). Ein Beispiel hierfür wäre: „Implementieren Sie den ifdef(VARIANTE1, ‘Quicksort’, ‘Mergesort’)-Algorithmus“ (Krinke, Störzer und Zeller 2002). Zwar ist dieses Konzept der Aufgabenindividualisierung sehr mächtig, jedoch kann das Erstellen von Aufgaben auf diesem Wege schnell sehr komplex und aufgrund der Anweisungen auch unübersichtlich werden. Im Kontext der Programmieraufgaben bietet sich außerdem das Individualisieren von Klassennamen, Attributen oder Beziehungen an. Dafür bietet ein Makromechanismus allerdings keine dedizierten Funktionalitäten. Natürlich könnte dies durch jeweils eigene Makros realisiert werden, das dafür nötige Know-how sollte jedoch nicht unterschätzt werden.

4.2 Kommunikation

In Kapitel 4.1 wurden einige Werkzeuge zur Automatisierung und Individualisierung von praktischen Aufgaben vorgestellt. Auch wenn diese Werkzeuge unterstützend wirken, spielen die Lehrenden weiterhin die zentrale Rolle. Durch die Automatisierung von Teilen der Lehre werden Zeitressourcen von Lehrenden verfügbar, welche beispielsweise in zusätzli-

che Betreuung von Lernenden investiert werden können. Diese Betreuungsleistung kann zum Teil durch Werkzeuge zur Kommunikation ausgebaut werden, weil diese einerseits schnelles Feedback bei Assessments und andererseits Active Learning in einem digitalen Rahmen ermöglichen.

Die Betreuung von Lernenden und damit die Kommunikation zwischen Lehrenden und Lernenden erfordert je nach Anliegen ein geeignetes Kommunikationsmedium. Dies ist auf die *Media Richness Theory* zurückzuführen. Laut dieser Theorie weisen verschiedene Kommunikationskanäle eine unterschiedliche Reichhaltigkeit an Informationen auf. Je mehr Reichhaltigkeit ein Kommunikationskanal aufweist, desto unwahrscheinlicher werden Missverständnisse und Mehrdeutigkeiten. Für komplexe Sachverhalte sollte infolgedessen auch ein Kommunikationskanal mit hoher Reichhaltigkeit wie beispielsweise die *Face-to-face* Kommunikation gewählt werden. Für simple Absprachen hingegen sollte ein Kommunikationskanal mit niedriger Reichhaltigkeit gewählt werden, denn ansonsten würde dies zu einer Verkomplizierung des Sachverhaltes und zu weniger effizienter Kommunikation führen (vgl. Daft und Lengel 1984). Dies wird durch Abbildung 2 verdeutlicht. In den folgenden zwei Kapiteln wird zum einen auf herkömmliche Kommunikationskanäle in der Lehre eingegangen und zum anderen aufgezeigt, welche Rolle moderne *Instant-Messenger* in der Lehre einnehmen könnten.

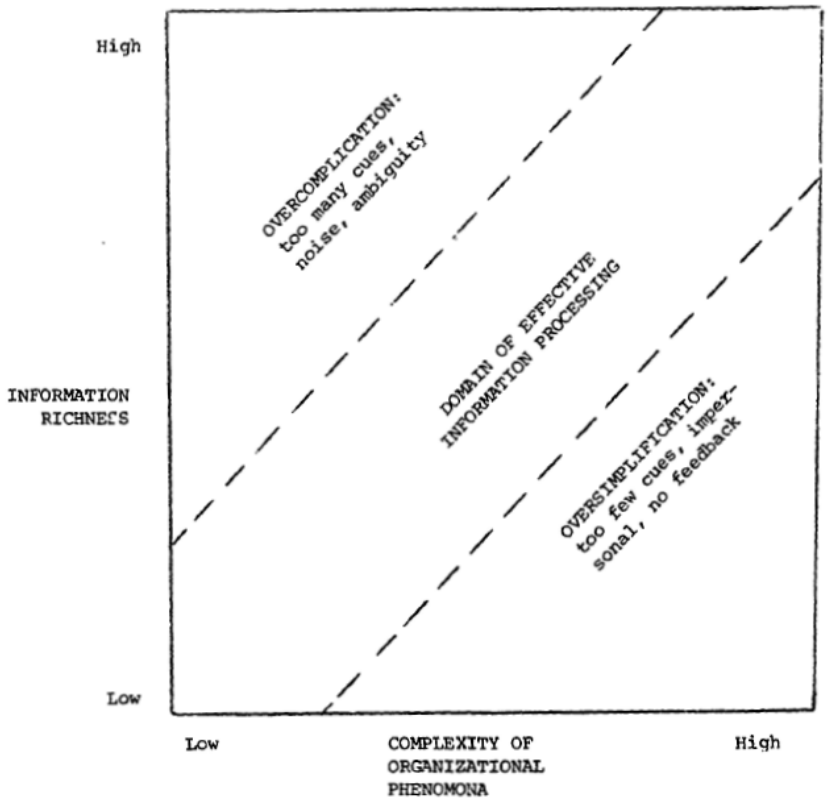

Abbildung 2: Bereich effektiver Kommunikation (Daft und Lengel 1984)

4.2.1 Kommunikationskanäle in der Lehre

In der Lehre sehr weit verbreitete Kommunikationskanäle sind die Face-to-face Kommunikation und die Kommunikation über E-Mails. Wie oben beschrieben, ist je nach Gegenstand der Kommunikation ein bestimmter Kommunikationskanal besser geeignet als ein anderer. Im Folgenden werden die Charakteristika der Kommunikationskanäle Face-to-face und E-Mail thematisiert.

E-Mail

Die Reichhaltigkeit von Informationen, welche über E-Mails vermittelt werden, ist niedrig bis mittelmäßig einzuordnen. Durch die Kommunikation erhaltene Hinweise beschränken sich auf das Geschriebene. Dies kann zu Mehrdeutigkeiten führen, welche aufgrund des langsamen Feedbacks über E-Mails schlecht aufgelöst werden können. Infolgedessen kann dies dazu führen, dass komplexe Problemstellungen simplifiziert werden. Wenn der Gegenstand der Kommunikation allerdings simpel und eindeutig ist, so ist dieser Kommunikationskanal sehr effizient und sollte in einem solchen Fall bevorzugt werden (vgl. Daft und Lengel 1984).

Face-to-face

Die Kommunikation über Face-to-face weist die höchste Reichhaltigkeit an Informationen auf. Im Gegensatz zur Kommunikation über E-Mails ist sofortiges Feedback durch den Kommunikationspartner möglich. Somit kann das eigene Verständnis in Bezug auf den Gegenstand der Kommunikation abgeklärt und angepasst werden. Anzumerken ist, dass das Feedback hierbei nur dann sofortig ist, wenn nicht vorher erst eine Terminabsprache für ein Treffen stattfinden muss. Außerdem findet diese Art der Kommunikation auf verschiedenen Ebenen statt. Erhaltene Hinweise können ihren Ursprung nicht nur im Gesagten, sondern auch in der Körpersprache, der Mimik oder in der Stimmlage haben. Jedoch können jene Hinweise widersprüchlich wirken oder vom Eigentlichen ablenken, was dazu führt, dass dieser Kommunikationskanal ineffizient sein kann, wenn es um einfache Problemstellungen geht. Infolgedessen sollte dieser Kommunikationskanal bevorzugt werden, wenn Anliegen komplex und ungewiss sind, sodass Mehrdeutigkeiten entstehen können (vgl. Daft und Lengel 1984).

4.2.2 Discord

Eine moderne Art zu kommunizieren sind *Instant-Messenger*. Diese erlauben das Verschicken und Empfangen von Nachrichten in nahezu Echtzeit. Moderne Instant-Messenger wie beispielsweise *Discord* erlauben neben dem Versenden von Textnachrichten außerdem Voicechats und Videochats. Discord ermöglicht somit die Kommunikation über verschiedene Kanäle, welche jeweils eine andere Reichhaltigkeit an Informationen aufweisen. Die Informationsreichhaltigkeit von Textnachrichten ist dabei etwas höher einzuschätzen als die von E-Mails, da schnelleres Feedback möglich ist. Dies ist dadurch begründet, dass Textnachrichten bei Instant-Messengern sofort zugestellt werden und nicht erst abgerufen werden müssen, wie es bei E-Mails der Fall ist. Wenn komplexere Anliegen kommuniziert werden müssen, können Voicechats oder Videochats Abhilfe schaffen. Allerdings ist anzumerken, dass diese immer noch weniger Reichhaltigkeit an Informationen aufweisen als die Face-to-face Kommunikation. Um der gesamten Kommunikation innerhalb eines Discord-Servers Struktur zu verleihen, können konfigurierbare Channel genutzt werden. Außerdem kann der Zugang auf besagte Channel mithilfe eines Gruppen-Rechtesystems auf konfigurierte Gruppen beschränkt werden. Wer sich zusätzliche Funktionalitäten wünscht, kann diese über sogenannte *Bots* integrieren (vgl. *Dein Ort zum Reden* o.D.).

Discord in der Lehre

Das Integrieren von einem Instant-Messenger wie beispielsweise Discord in die Lehre soll keinesfalls herkömmliche Kommunikationskanäle komplett ersetzen. Vor allem die Face-to-face Kommunikation ist und bleibt ein wichtiger Bestandteil der Lehre. Zwar können

einige Absprachen, welche normalerweise über E-Mails stattfinden auch über Discord kommuniziert werden, jedoch sollte der Kommunikationsweg über E-Mails weiterhin möglich sein, insbesondere dann, wenn einzelne Lernende nicht bereit sind, Discord zu installieren oder zu nutzen. Letzteres ist allerdings sehr unwahrscheinlich, weil Discord vor allem bei der jüngeren Generation sehr weit verbreitet ist.

In Abgrenzung zu herkömmlichen Kommunikationswegen in der Lehre ermöglichen Instant-Messenger wie Discord zudem das Abbilden von virtuellen Klassenräumen, welche über festgelegte wöchentliche Zeitslots hinausgehen. Innerhalb dieser virtuellen Klassenräume können Lernende schnelles Feedback von anderen Lernenden oder auch Lehrenden erhalten. Außerdem können diverse Problemstellungen auch außerhalb von regulären Präsenzterminen identifiziert und diskutiert werden. Sowohl das schnellere Feedback als auch die Übersicht über diskutierte Problemstellungen innerhalb der Channel beeinflussen das Assessment (siehe Kapitel 2.1.4) innerhalb eines Kurses in positiver Weise. Schlussendlich ist zu erwähnen, dass Discord aufgrund von bestimmten Datei-Upload-Limits und Nutzer-Limits innerhalb von Videochats bestimmte Werkzeuge in der Lehre nicht ersetzen kann (vgl. De Kok 2020; Skains 2020). So muss für die Verwaltung von Kursmaterial weiterhin auf Dienste wie beispielsweise Ilias (siehe Kapitel 4.1.2) und für Online-Vorlesungen weiterhin auf Dienste wie beispielsweise *Zoom* zurückgegriffen werden.

5 Gestaltung und Organisation von Praktika

In diesem Kapitel werden Ansätze und empfohlene Konzepte zur Gestaltung und Organisation von modernen Programmierpraktika vorgestellt. Diese ergeben sich aus recherchierter Literatur und aus durchgeführten Experteninterviews. Infolgedessen basieren die folgenden Designentscheidungen auf den erhobenen Anforderungen aus den Kapiteln 2.1.5, 2.2.4 und 3.2.5.

5.1 Aufbau und Ablauf

Die Gestaltung und der Ablauf eines jeden Kurses in der Lehre sind abhängig von dessen Lernzielen. Dennoch werden in diesem Kapitel einige Empfehlungen zur Gestaltung von praxisorientierten Kursen mit Programmieranteil diskutiert. Im Rahmen der Lehre von modernen Programmieransätzen ist gerade der Programmieranteil von großer Bedeutung. Mit Blick auf die Revised Bloom's Taxonomy (siehe Kapitel 2.1.3) kann nur durch das Schreiben von Programmcode in umfangreichem Ausmaß die sechste Stufe der kognitiven Prozesse *Creating* in den verschiedenen Wissensdimensionen erreicht werden.

Außerdem ist es sinnvoll, dass jene Aufgaben in Form von vollständigen Projekten auf einer Plattform wie beispielsweise Gitlab (siehe Kapitel 4.1.3) zur Verfügung gestellt werden. Im Gegensatz zu dem Lösen von Aufgaben durch das Hochladen von Programmteilen auf beispielsweise Ilias (siehe Kapitel 4.1.2), werden Lernende somit früh mit der Struktur von eben solchen Projekten vertraut gemacht. Zugleich können erste Erfahrungen mit Werkzeugen wie beispielsweise *Build-Management-Tools* oder Git gesammelt werden. Laut Bente (2020, Interview, siehe Anhang 2.2.3, Z. 460-470) ist die Plattform Gitlab außerdem eine gute Wahl, weil Gitlab Lehrenden viel Kontrolle und Änderungsmöglichkeiten innerhalb der Kette von benutzten Werkzeugen ermöglicht.

Besagte Programmieraufgaben können einerseits Teil von Workshops sein, welche nach dem Prinzip Active-Learning (siehe Kapitel 2.1.1) gestaltet werden. Innerhalb dieser Workshops können auf Inhaltsimpulse kleinere Aufgaben folgen, welche anschließend bearbeitet werden. Auch wenn diese später nicht abgegeben werden, können die Ergebnisse im Anschluss verglichen und diskutiert werden. Andererseits können diese Programmieraufgaben ebenso Teil von Praktika sein, welche Aufgaben umfassen, die von Lernenden gelöst und später abgegeben werden sollen. Darüber hinaus können diese Aufgaben individualisiert werden. Ein hoher Individualisierungsgrad kann gewählt werden, wenn das Kopieren von Lösungen stark erschwert werden soll. Dahingegen wird ein vergleichsweise geringer Individualisierungsgrad dann gewählt, wenn Gruppenarbeit mit einzelnen Ab-

gaben ermöglicht werden soll. Besagte Gruppenarbeit sollte allerdings weniger stark im Fokus stehen, wenn es sich um einen Programmierkurs für Anfänger handelt. In einem solchen Szenario ist Gruppenarbeit weniger förderlich, weil grundlegende Konzepte thematisiert werden, für die jeder Lernende ein eigenes Verständnis entwickeln muss. An dieser Stelle kann der Ansatz Pair-Programming eingesetzt werden, damit Lernende ihr Verständnis von Programmcode mit dem Verständnis anderer abgleichen. Voraussetzung dafür ist allerdings, dass sich Lernende an die Prinzipien und Vorgaben des Ansatzes halten (siehe Kapitel 3.2.4).

Zusätzlich zu dem Feedback von Betreuern können bei Praktikumsaufgaben automatische Tests eingesetzt werden, um schnell erstes Feedback zu generieren. Dieses Feedback soll das Feedback der Betreuer keinesfalls ersetzen. Kohls (2020, Interview, siehe Anhang 2.2.2, Z. 471-480) bezeichnet das Feedback durch automatische Tests als eine Art *First-Level-Support*, welcher unter anderem Betreuer entlastet, damit sich diese anschließend aufgrund freier Zeitressourcen gezielter bestimmten Problemen annehmen können. Reitano (2021, Interview, siehe Anhang 2.2.4, Z. 327-336) betont, dass menschliches Feedback gerade da wichtig ist, wo es weniger um erfüllte funktionale Anforderungen, sondern mehr um Design und Architektur geht.

5.1.1 Motivation der Lernenden

Ein wichtiger Einflussfaktor, welcher den Lernerfolg einzelner Lernender bestimmt, ist die Motivation dieser. Diese lässt sich wie in Kapitel 2.1.2 beschrieben in intrinsische und extrinsische Motivation unterteilen. Lernprozesse sind effizienter, wenn ein Lernender intrinsisch motiviert ist. Da dies allerdings nicht immer der Fall ist, wird häufig extrinsisch über externe Regulierungen motiviert. Auf diese Art der extrinsischen Motivation sollte allerdings so wenig wie möglich zurückgegriffen werden, weil diese die intrinsische Motivation negativ beeinflussen kann.

Reitano (2021, Interview, siehe Anhang 2.2.4, Z. 176-200) fügt außerdem an, dass Lernende zu anfangs meist intrinsisch motiviert sind, diese Motivation jedoch von bestimmten Faktoren negativ beeinflusst werden kann. Zum einen sind Lernende weniger intrinsisch motiviert, wenn diese von den Lehrenden nicht ernst genommen werden. Zum anderen darf den Lernenden kein widersprüchliches Feedback gegeben werden. Lehrende dürfen also keine Unwahrheiten vermitteln, welche sich durch eigene Recherchen von Lernenden als falsch herausstellen würden. Zusätzlich sollte vermieden werden, dass widersprüchliche Meinungen von mehreren Betreuern Lernende verwirren. Als mögliche Lösung schlägt Reitano dabei zusätzlich zur Beratungsleistung automatische Tests vor, die nur jene Wahrheit

vermitteln, welche im Programmcode der Tests selbst liegt.

Praktika setzen sich meist aus mehreren Meilensteinen zusammen, welche nacheinander von Lernenden bearbeitet und bestanden werden müssen, um zur Klausur zugelassen zu werden. Dahingegen verfolgt Kohls (2020, Interview, siehe Anhang 2.2.2, Z. 289-335) den Ansatz, dass Meilensteine unabhängig voneinander bestanden werden können und zudem sogar mehrmals im Semester angeboten werden. Dies führt dazu, dass ein Lernender nicht bis an den Anfang des Praktikums zurückgeworfen wird, wenn dieser einen Meilenstein nicht bestanden hat. So entsteht weniger Stress und es wird weniger intrinsische Motivation eingebüßt. Der Fokus steht in diesem Modell mehr auf dem Erreichten und weniger auf dem Druck, Meilensteine bestehen zu müssen.

Zusätzlich wirkt sich eine hohe Verbindlichkeit sowohl unter den Lehrenden und Lernenden als auch unter einzelnen Lernenden positiv auf die intrinsische Motivation dieser aus. Diese Verbindlichkeit könnte beispielsweise durch Lehreinheiten in Blöcken mit fester Tagesstruktur gefördert werden (vgl. Bente 2020, Interview, siehe Anhang 2.2.3, Z. 262-278).

5.1.2 Betreuung der Lernenden

Ein wichtiger Bestandteil der praxisorientierten Lehre bildet die Betreuung der Lernenden. Diese kann auf verschiedenen Wegen erfolgen. In Kapitel 4.2.2 wurden neben den klassischen Kommunikationskanälen in der Lehre Instant-Messenger und dessen Vorteile thematisiert. Zusammenfassend kann festgehalten werden, dass Instant-Messenger wie beispielsweise Discord eine flexiblere Beratung zu gewünschten Zeitpunkten ermöglichen und somit Lernende schneller Feedback erhalten können.

Auch wenn die Face-to-face Kommunikation oft der Kommunikationskanal der Wahl ist, kann zusätzlich innerhalb Discord eine Art virtueller Klassenraum entstehen. Wenn in diesem ein aktiver Austausch zwischen Lernenden und Lehrenden und zwischen einzelnen Lernenden stattfindet, kann somit die Verbindlichkeit zwischen allen Teilnehmern gesteigert werden. Wie bereits erwähnt kann sich diese Verbindlichkeit positiv auf die intrinsische Motivation der Lernenden auswirken. Bente (2020, Interview, siehe Anhang 2.2.3, Z. 476-498) ist außerdem der Meinung, dass Discord eine leichtgewichtige und leicht verständliche Kommunikationsplattform ist, mit der sich eine hybride Lehre aufsetzen lässt, welche Teile der Lehreinheiten mit Discord-Unterstützung umsetzt.

Ein Problem von Instant-Messengern ist, dass Lernende bei der Nutzung dieser oft die

Erwartung haben, dass Nachrichten sofort beantwortet werden. Daher sollte es einem Lehrenden möglich sein, mitzuteilen, ob dieser gerade erreichbar ist oder nicht. In diesem Fall sollten Nachrichten für einen späteren Zeitpunkt gesammelt werden. Zusätzlich wird mit Instant-Messengern immer auch eine Datenschutzfrage verbunden, vor allem dann, wenn persönliche Daten wie beispielsweise Telefonnummern weitergegeben werden müssen (vgl. Kohls 2020, Interview, siehe Anhang 2.2.2, Z. 500-518). Auch wenn im Fall von Discord kein eigenes Hosten eines Servers möglich ist, sind Nutzer innerhalb Discord anonym unterwegs und müssen bei einer Registrierung lediglich eine E-Mail angeben, welche dritten nicht sichtbar gemacht wird.

Innerhalb der Experteninterviews wurde außerdem der Instant-Messenger *Slack* erwähnt, welcher zusätzlich Integration zu Building-Tools mitbringt (vgl. Interviewter 1 2020, Interview, siehe Anhang 2.2.1, Z. 378-396). Schlussendlich ist anzumerken, dass eine hochschulweite Lösung von Vorteil ist, um eine hohe aktive Teilnahme auf Seiten der Lernenden zu erreichen, denn auf diese Weise werden zusätzliche Hürden für die Lernenden vermieden.

5.2 Integration moderner Konzepte der Programmierung

In Kapitel 2.2 wurden drei wesentliche Konzepte der Programmierung vorgestellt. Jedes dieser Konzepte lässt sich auf unterschiedliche Weise in die praxisorientierte Lehre von modernen Programmieransätzen integrieren. Im Folgenden wird auf die Konzepte Domain Driven Design, Clean Code und Test Driven Development eingegangen.

5.2.1 Domain Driven Design

Um Domain Driven Design lehren zu können, muss ein gewisses Verständnis für grundlegende Konzepte der Programmierung vorausgesetzt werden. In jedem Fall sollten Lernende fähig sein, sauberen Programmcode schreiben zu können, welcher gängigen Qualitätskriterien entspricht. Außerdem ist es laut Reitano (2021, Interview, siehe Anhang 2.2.4, Z. 119-131) wichtig, dass Lernende frühzeitig saubere Objektorientierung beherrschen, weil diese die Basis für Domain Driven Design bildet. Darüber hinaus muss früh gelehrt werden, dass bei der Modellierung von Software nicht nur in Daten, sondern auch in Verhalten gedacht werden muss.

Auch wenn diese Voraussetzungen gegeben sind, ist Domain Driven Design dennoch ein sehr komplexer Ansatz. Laut Bente (2020, Interview, siehe Anhang 2.2.3, Z. 171-195) lässt sich das Tactical Design leichter in die Lehre integrieren als das Strategic Design. Vor allem für letzteres sind in der Lehre oft nicht die nötigen Kapazitäten vorhanden.

Häufig fallen iterative Vorgehensweisen wegen Zeitmangel weg, obwohl diese ein charakteristisches Merkmal von Domain Driven Design sind. Außerdem steht in der Lehre meist keine Fachseite zur Verfügung, welche aufkommende Fragen während der Entwicklung beantwortet. Folglich würde Strategic Design eher in einem Masterstudiengang thematisiert werden. Jedoch bietet es sich auch hier aufgrund des Zeitmangels an, den Bounded Context Schnitt vorzugeben.

Allerdings lassen sich im Rahmen des Tactical Designs die Themenbereiche Building Blocks und Ubiquitous language gut in die Lehre integrieren. Glossare in Form von strukturierten Tabellen können sogar einer ersten automatischen Überprüfung unterzogen werden, um Lernenden schnell ein erstes Feedback bezüglich dieser zur Verfügung zu stellen. Wichtig bei der Erstellung von Aufgaben im Bereich des Tactical Designs ist dabei, dass Aufgabentexte ausführlich genug formuliert werden, sodass Lernende genügend Hinweise erhalten, um fundierte Designentscheidungen treffen zu können (vgl. Bente 2020, Interview, siehe Anhang 2.2.3, Z. 171-195).

5.2.2 Clean Code

Die Relevanz von Clean Code und damit verbundenen Prinzipien und Richtlinien wird bereits in Kapitel 2.2.2 thematisiert. Es wird unter anderem betont, dass Lernende mit schlechten Programmcode in Berührung kommen müssen, um ein Gefühl dafür zu bekommen, was sauberen Programmcode ausmacht. Dem schließt sich Kohls (2020, Interview, siehe Anhang 2.2.2, Z. 134-150) an und betont, dass Clean Code Prinzipien nicht einfach im Frontalunterricht gelehrt werden können, sondern dass Lernende durch Feedback auf Basis von Programmcode aufgezeigt bekommen müssen, warum bestimmte Zeilen dieses Programmcodes unsauber sind. Um die Bedeutung mancher Prinzipien nachvollziehen zu können, müssen Lernende erleben, in welche Probleme man läuft, wenn jene Prinzipien nicht eingehalten werden.

Besagtes Feedback kann auf unterschiedliche Weisen gegeben werden. Zum einen kann Clean Code in der Lehre explizit zum Thema gemacht werden. Allerdings sollte dies nicht in reinem Frontalunterricht passieren, sondern beispielsweise über die kritische Betrachtung und Analyse alter Projekte von Lernenden (vgl. Bente 2020, Interview, siehe Anhang 2.2.3, Z. 113-129). Zum anderen kann Clean Code auch in bestehenden Kursen mit Programmieranteil integriert werden, indem auf unsaubere Fragmente von Programmcode aufmerksam gemacht wird. Dies könnte zusätzlich um automatische Mechanismen angereichert werden, welche von Lernenden eingereichte Lösungen in Hinblick auf bestimmte Clean Code Richtlinien überprüfen (vgl. Bente 2020, Interview, siehe Anhang

2.2.3, Z. 136-150). Hierfür existieren mehrere technische Möglichkeiten wie beispielsweise die Überprüfung mittels Unit-Tests oder eine statische Code Analyse durch SonarQube.

Allerdings sind bestimmte Richtlinien wie beispielsweise eine aussagekräftige Benennung von Variablen schwer automatisch prüfbar. In diesem Fall könnten Peer-Review Verfahren (siehe Kapitel 2.1.4) genutzt werden, um Code bezüglich der Verständlichkeit auf Basis gegebener Richtlinien gegenseitig von Lernenden bewerten zu lassen (vgl. Reitano 2021, Interview, siehe Anhang 2.2.4, Z. 103-111).

Schlussendlich muss betont werden, dass Lernende Clean Code Prinzipien umso besser verinnerlichen können, je häufiger diese gründlich und nachvollziehbar thematisiert werden. Daher bietet es sich an, Clean Code früh, eventuell sogar in einem extra dafür vorgesehenen Kurs, zum Thema zu machen. Somit werden Lernende rechtzeitig darauf aufmerksam gemacht, dass Clean Code in der Programmierung eine äußerst wichtige Rolle einnimmt. Über diesen Kurs hinaus sollten Folgekurse auf gelernten Prinzipien aufbauen und diese fordern. Vorhin beschriebene automatische Mechanismen könnten dann dazu genutzt werden, Lernende daran zu hindern, unsauberen Programmcode abzugeben. Infolgedessen würden Lernende dazu bewegt werden, sich mit ihrem Programmcode und gegebenen Richtlinien auseinanderzusetzen. Letztendlich sollten Betreuer jener Kurse fähig sein, die Fragen Lernender diesbezüglich qualitativ zu beantworten.

5.2.3 Test Driven Development

In Kapitel 2.2.3 wird der Ansatz Test Driven Development bereits zusammen mit dem damit einhergehenden iterativen Vorgehen vermittelt. Im weiteren Verlauf soll diskutiert werden, inwiefern der Ansatz Test Driven Development in die praxisorientierte Lehre von modernen Programmieransätzen integriert werden kann.

Im Folgenden werden die Vorteile von Test Driven Development aufgezeigt (vgl. George und Williams 2004, S. 338f):

1. Wenn Programmcode gewartet wird, muss viel Zeit für das Verständnis des Programmcodes aufgebracht werden. Test Driven Development hilft während dieser Phase durch das Schreiben von Tests, welche infolgedessen immer aktuell sind

2. Test Driven Development liefert Feedback während der Implementierung, weshalb Debugging Phasen zur Fehlersuche verkürzt werden

3. Bestehende Tests erlauben effizientes *Regression Testing*. Hierbei können neu entstandene Fehler durch neue Modifikationen schnell identifiziert werden

4. Durch Test Driven Development werden viele Fehler im Programmcode schnell entdeckt, wodurch hohe Kosten durch erst spät gefundene Fehler reduziert werden

Im Anschluss an die Vorteile werden die Nachteile von Test Driven Development aufgezeigt (vgl. George und Williams 2004, S. 338):

1. Durch Test Driven Development werden die drei Phasen Design, Implementierung und Testen leicht vermischt, sodass die Design Phase mehr die Implementierung im Detail berücksichtigt und weniger die Software als Ganzes umfasst
2. Bestimmte Teile eines Programms sind schwer zu testen (z.B. GUIs)
3. Es muss Zeit dafür aufgewendet werden, Mock Objects zu erstellen
4. Oft ersetzen Tests formale Dokumentation, sodass wichtige Gedanken für Designentscheidungen verloren gehen
5. Test Driven Development setzt stetiges *Refactoring* voraus, um Code Verständnis und Komplexität unter Kontrolle zu bekommen
6. Entwickler müssen ein hohes Verständnis für Programmcode entwickelt haben, um Tests für Programmcode zu schreiben, welcher schwer zu testen ist

Die Vorteile sprechen für sich. Allerdings wird nach genauerer Betrachtung der Nachteile deutlich, dass die Nachteile eins, vier und sechs vor allem dann zum Tragen kommen, wenn Entwickler nicht genügend Erfahrungen im Bereich der Programmierung oder der Anwendung des Ansatzes selbst vorweisen können. Deshalb ist davon abzuraten, den Ansatz Test Driven Development früh in der praxisorientierten Lehre umfassend zu thematisieren. Allerdings sollten Lernende rechtzeitig mit grundlegenden Prinzipien, welche der Ansatz fordert, konfrontiert werden. Infolgedessen würde es helfen, wenn Lernende im Rahmen von praxisorientierten Programmierkursen lernen, gegen Tests zu entwickeln, welche vorab durch Lehrende vorgegeben werden. Dies ist dadurch begründet, dass das Entwickeln gegen Tests ein wesentlicher Bestandteil des Test Driven Developments ist. Zumindest die Schritte zwei bis vier des in Kapitel 2.2.3 vorgestellten Vorgehens könnten somit abgedeckt werden. Zugleich werden Lernende mit dem Konzept von Mock Objects vertraut gemacht, welche gegebenenfalls durch Lehrende definiert werden. Durch das frühe Konfrontieren mit diesen Prinzipien kann ein grundlegendes Verständnis geschaffen werden, welches es ermöglicht, Test Driven Development in vollem Ausmaß zu einem späteren Zeitpunkt thematisieren zu können. Schlussendlich ist anzumerken, dass für die Lehre von Test Driven Development genügend zeitliche Kapazitäten einzuplanen sind, wenn das für den Ansatz typische iterative Vorgehen berücksichtigt werden soll.

6 Werkzeuge zur Automatisierung und Individualisierung

In diesem Kapitel werden ein Konzept und passende Werkzeuge vorgestellt, mit dessen Hilfe sich Praktika mit individualisierten Programmieraufgaben realisieren lassen, welche bis zu einem bestimmten Grad automatisch getestet werden können. Die Grundlage bilden dabei Gitlab (siehe Kapitel 4.1.3) und das *Divekit* (steht für *Digital individualized exercises*), welches Aufgaben optional individualisiert und anschließend auf Gitlab verteilt. Sowohl das vorgestellte Testkonzept, als auch das Divekit wurden dabei entwickelt, um den in den Kapiteln 2.1.5, 2.2.4 und 3.2.5 aufgelisteten Anforderungen gerecht zu werden. Bevor näher auf die einzelnen Bereiche der Automatisierung und Individualisierung eingegangen wird, soll Darstellung 3 einen Überblick über das Konzept mitsamt relevanten Komponenten und Werkzeugen verschaffen.

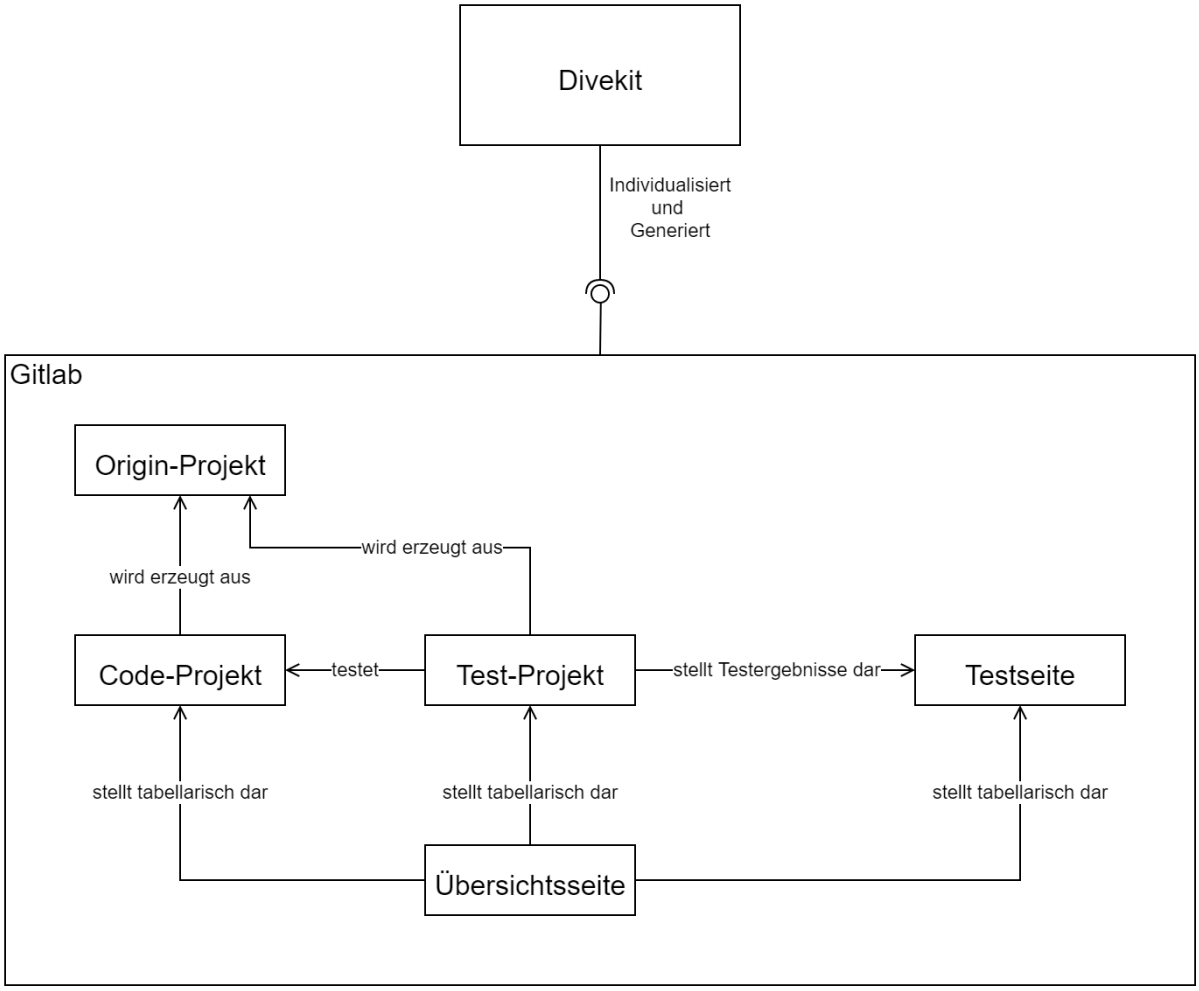

Abbildung 3: Übersicht über relevante Komponenten und Werkzeuge

Die folgenden Komponenten und Werkzeuge sind Bestandteile des Gesamtkonzeptes:

- Auf *Gitlab* werden sämtliche Aufgaben verwaltet. Durch Hochladen von Lösungen auf Gitlab können Lernende Aufgaben bearbeiten. Über Gitlab CI werden außerdem automatisch Tests ausgeführt. Bestandteile eines Praktikums können in Form von Gitlab Gruppen verwaltet werden, wobei eine Gruppe einem Meilenstein innerhalb des Praktikums entsprechen würde.

- Bei dem *Origin-Projekt* handelt es sich um ein normales Projekt, dessen Struktur von der im Praktikum thematisierten Programmiersprache abhängig ist. Neben einer Aufgabenstellung beinhaltet das Projekt außerdem Programmfragmente, welche bei der Aufgabenbearbeitung vorgegeben werden und Tests, welche später automatisch ausgeführt werden. Zusätzlich können in diesem Projekt Lösungen für Aufgaben und Konfigurationsdateien für beteiligte Werkzeuge enthalten sein. Aus diesem Projekt werden später für jeden Lernenden jeweils ein Code-Projekt und ein Test-Projekt generiert. Bei diesem Vorgang werden Dateien gefiltert, welche nichts in den jeweiligen Projekten zu suchen haben (z.B.: Lösungen).

- Das *Code-Projekt* wird aus dem Origin-Projekt generiert. Es enthält alles, was für die Aufgabenbearbeitung durch einen Lernenden benötigt wird. Wenn ein Lernender eine Aufgabe fertig bearbeitet hat oder einen Zwischenstand sichern will, so wird dieser auf das Code-Projekt mittels Git hochgeladen.

- Das *Test-Projekt* enthält meist alle Dateien, welche auch das Code-Projekt enthält, jedoch können hier zusätzliche Dateien abgelegt werden. Hauptaufgabe des Test-Projekts ist deshalb das Kapseln von Dateien, welche nicht für Lernende bestimmt sind (z.B. versteckte Tests). Außerdem führt das Test-Projekt automatische Tests über Gitlab CI aus und generiert anschließend eine Testseite.

- Aus Gründen der Modularisierung wird sämtlicher wiederverwendbarer Programmcode von automatischen Tests innerhalb einer *Test-Library* gepflegt. Diese ist programmiersprachenabhängig und wird von dem Code-Projekt und dem Test-Projekt referenziert.

- Die *Testseite* wird von dem Test-Projekt für jeden Lernenden einzeln generiert und zeigt jeweils die Testergebnisse eines Lernenden in Form einer statischen Seite an.

- Da jeder Lernende pro Meilenstein innerhalb von Gitlab ein eigenes Code-Projekt, Test-Projekt und eine eigene Testseite hat, wird eine *Übersichtsseite* erzeugt, welche jeden Lernenden dementsprechend zuordnet.

- Innerhalb Gitlab existieren keine Automatismen, welche für jeden Lernenden die passenden Projekte mitsamt Aufgaben erzeugen können. Um diesen Anforderungen gerecht zu werden, wurde das *Divekit* entwickelt. Hierbei handelt es sich um ein lokal ausführbares Programm, welches in Typescript entwickelt wurde. Dieses erzeugt auf Basis von Konfigurationsdateien über die Gitlab API für jeden Lernenden die passenden Projekte und generiert auf dieser Basis die Übersichtsseite. Zusätzlich sind Mechanismen zur Aufgabenindividualisierung implementiert, welche optional genutzt werden können.

Die wichtigste Rolle bei diesem Konzept spielen natürlich die Lernenden und Lehrenden, welche mit den oben genannten Komponenten interagieren. Wenn ein neuer Praktikumsmeilenstein ansteht, erstellt ein Lehrender eine neue Aufgabe in Form eines Origin-Projekts. Dabei nutzt er das Divekit zum automatischen Generieren der Projekte für die Lernenden. Um von dem Divekit bereitgestellte Automatismen vorab zu testen, kann das eigene Dateisystem anstelle der Gitlab API angesprochen werden. Nachdem die Projekte für die Lernenden erzeugt wurden, können diese die Aufgaben, welche sich im Code-Projekt befinden, bearbeiten und erhalten nach jedem Hochladevorgang schnelles Feedback über automatische Tests, welche ausgeführt und dessen Ergebnisse auf der Testseite zusammengefasst werden. Während und auch nach einem Meilenstein können Lehrende über die Übersichtsseite Projekte den einzelnen Lernenden zuordnen und zudem auf der Testseite eines jeden einsehen, wie viele Tests erfolgreich durchgelaufen sind.

6.1 Aufgabenüberprüfung

Vorgegebene automatische Tests in Programmierpraktika ermöglichen einen First-Level-Support für Lernende, welchen durch die Tests schnelles Feedback zuteilwird. Besagtes Feedback hat sowohl formativen als auch summativen Charakter (siehe Kapitel 2.1.4). Zugleich trainieren Lernende, geschriebenen Programmcode ihrerseits an den Anforderungen vorgegebener Tests auszurichten. Im Folgenden wird der Teil aus der Übersicht des Gesamtkonzepts beschrieben, welcher automatische Aufgabenüberprüfung mittels Tests innerhalb Gitlab realisiert. Das Konzept lässt sich auch auf andere Plattformen übertragen, wenn diese ähnliche Funktionalitäten wie Gitlab aufweisen.

6.1.1 Code- und Test-Projekt

Sowohl das Code-Projekt als auch das Test-Projekt werden aus dem Origin-Projekt für jeden Lernenden erzeugt. Im Code-Projekt laden Lernende ihre Lösungen hoch, welche anschließend von dem Test-Projekt mittels automatischer Tests überprüft werden. Es ist anzumerken, dass Lernende nur Zugriff auf das Code-Projekt haben, jedoch nicht auf

das jeweilige Test-Projekt. Dabei sind vier Gründe zu nennen, warum es sich um zwei separate Projekte handeln muss und die zwei Projekte nicht in einem einzigen Projekt vereint werden können:

- Da Aufgaben optional individualisiert werden, variieren somit natürlich auch die möglichen Lösungen. Falls Lehrende während der Betreuung oder aus einem anderen Grund auf vorher definierte mögliche Lösungen zugreifen wollen, so können die Lösungen somit aufgrund der Variationen nicht zentral gespeichert werden. Im Test-Projekt kann mitsamt der Aufgabenstellung jeweils eine passende individualisierte Lösung generiert werden, welche für die Lernenden nicht einsehbar ist.
- Wenn Aufgaben eine manuelle Korrektur erfordern, so kann Feedback von Lehrenden in Dateien innerhalb des Test-Projekts abgelegt werden. Diese Dateien sind ebenfalls nicht von Lernenden einsehbar und können von Tests während der Generierung der Testseite ausgelesen werden. Somit kann Feedback von Lehrenden mitsamt anderer Testergebnisse auf der Testseite dargestellt werden.
- Durch fehlenden Zugriff auf das Test-Projekt auf Seiten der Lernenden können dort Tests gespeichert werden, dessen Programmcode nicht einsehbar sein soll. Lediglich das Ergebnis des Tests ist auf der Testseite einsehbar.
- Da die Ergebnisse von automatischen Tests nicht verfälscht werden dürfen, sollten Lernende nicht die Möglichkeit haben, den Programmcode der Tests zu modifizieren, welche schlussendlich auf der Testseite angezeigt werden. Aus diesem Grund sind sämtliche Tests nicht nur in dem Code-Projekt zur schnellen lokalen Kontrolle gespeichert, sondern sind außerdem als Kopie im Test-Projekt enthalten. Modifiziert nun ein Lernender lokal den Programmcode eines Tests, so wird dies beim Generieren der Testseite ignoriert.

Auch wenn es sich bei dem Code- und Test-Projekt um zwei separate Projekte handelt, sind diese dennoch miteinander verknüpft. Diese Integration passiert mittels Gitlab CI Skripts und dazugehörigen hinterlegten Variablen. Im Folgenden werden die automatischen Abläufe beschrieben, welche nötig sind, um von hochgeladenen Lösungen bis hin zur Testseite zu gelangen:

- Mittels Git lädt ein Lernender Lösungen hoch auf das Code-Projekt.
- Durch das Hochladen wird die Gitlab CI Pipeline des Code-Projekts ausgelöst. Damit das Test-Projekt dies mitbekommt, löst die Pipeline des Code-Projekts die Pipeline des Test-Projekts aus.

- Das Test-Projekt kopiert sich nun die hochgeladenen Lösungen des Lernenden von dem Code-Projekt in einen eigenen Bereich. Bei diesem Vorgang werden nur relevante Dateien kopiert, sodass Lernende auf diese Weise keine wichtigen Dateien innerhalb des Test-Projekts überschreiben können.
- Anschließend führt das Test-Projekt alle darin enthaltenen Tests aus und generiert auf dieser Basis schlussendlich die Testseite.

6.1.2 Testseite

Die Testseite wird von dem Test-Projekt generiert und stellt die Testergebnisse von ausgeführten Tests dar. Bei der Testseite handelt es sich um eine statische Seite, welche von der Gitlab Instanz gehostet wird. Der Programmcode der Testseite wurde von jemand anderem als dem Verfasser dieser Arbeit während eines gemeinsam betreuten Kurses geschrieben. Abbildung 4 soll einen Einblick in ein mögliches Erscheinungsbild der Testseite vermitteln. Grün aufgeführte Tests sind erfolgreich durchgelaufen, rot aufgeführte Tests zeigen auf einen Klick hin eine Fehlermeldung an.

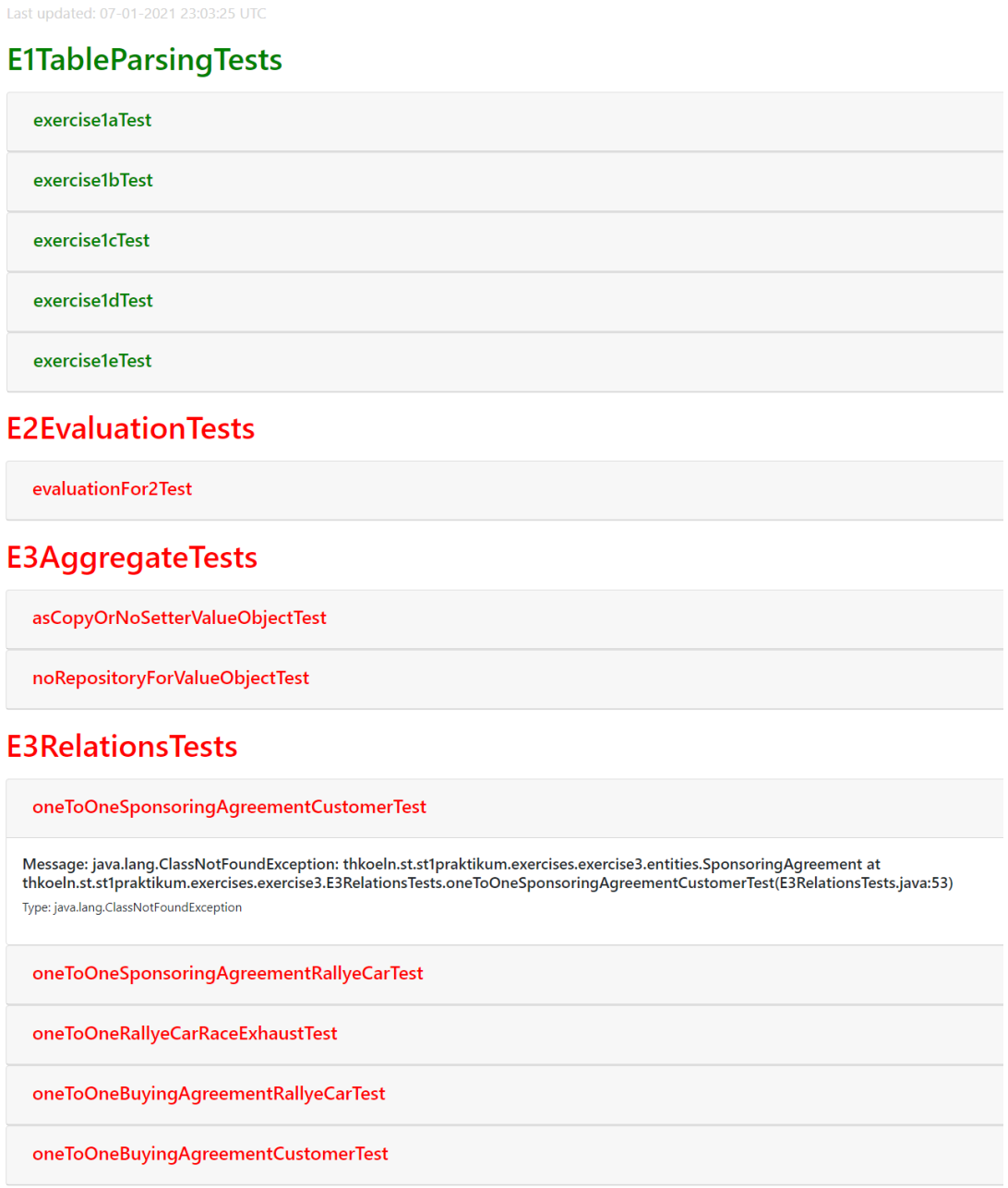

Abbildung 4: Mögliches Erscheinungsbild einer Testseite

6.1.3 Test-Library

Über mehrere Meilensteine innerhalb eines Praktikums oder sogar über mehrere Kurse hinweg kann es vorkommen, dass bestimmte automatische Tests wiederverwendet werden können. Aus diesem Grund bietet es sich an, diese an einem zentralen Ort in Form einer Test-Library zu verwalten. Als Beispiel hierfür können Tests aufgeführt werden, welche überprüfen, ob Beziehungen zwischen Objekten richtig implementiert wurden. Die Test-Library kann dann anschließend von dem Code- und Test-Projekt referenziert werden,

sodass innerhalb diesen eine breite initiale Auswahl an Tests zur Verfügung steht.

In diesem Kontext überprüfen Tests meist Programmcode, welcher noch nicht geschrieben wurde. Damit ein Projekt dennoch kompiliert, muss dieser Umstand dementsprechend beachtet werden. Einerseits können wie bei dem Test Driven Development (siehe Kapitel 2.2.3) vorab sogenannte Mock Objects definiert werden. Wenn die relevante Programmiersprache allerdings *Reflection* unterstützt, können somit geforderte Schnittstellen vorausgesetzt werden, welche noch nicht implementiert wurden. Der Programmcode der Tests ist natürlich programmiersprachenabhängig. Im Folgenden wird jedoch ein Beispiel anhand der Programmiersprache Java gezeigt, welches außerdem das Spring-Framework voraussetzt. In Listing 1 wird demonstriert, wie mithilfe von Reflection eine bestimmte Art von Beziehung zwischen zwei Objekten auf eine korrekte Implementierung getestet wird.

```
1  public void oneToOneTest(ObjectDescription parentObjectDescription,
2      ObjectDescription childObjectDescription) throws Exception {
3
4      // Create Parent & Child object
5      Object parentObject = objectBuilder
6          .buildObject(parentObjectDescription);
7      Object childObject = objectBuilder
8          .buildObject(childObjectDescription);
9
10     // Retrieve Methods
11     Method setRelationMethod = parentObject.getClass()
12         .getMethod(childObjectDescription.getSetToOne(),
13         childObject.getClass());
14     Method getRelationMethod = parentObject.getClass()
15         .getMethod(childObjectDescription.getGetToOne());
16
17     // Save parent with child as attribute and child to repository
18     setRelationMethod.invoke(parentObject, childObject);
19     oir.getRepository(childObjectDescription.getClassPath())
20         .save(childObject);
21     oir.getRepository(parentObjectDescription.getClassPath())
22         .save(parentObject);
23
24     // Retrieve parent with child as attribute from repository
25     Object retrievedParentObject = oir
26         .getRepository(parentObjectDescription.getClassPath())
27         .findAll().iterator().next();
28     Object retrievedChildObject = getRelationMethod
```

```
29          .invoke(retrievedParentObject);
30      assertEquals(childObject, retrievedChildObject);
31  }
```

Listing 1: Beispiel einer abstrakten Testmethode

Der in Listing 1 dargestellte Test ist sehr abstrakt geschrieben und kann folglich durch entsprechende Parametrisierung auf verschiedene Paare von Objekten angewendet werden. Damit die Parameterliste der Testmethode allerdings nicht zu lang wird, werden sogenannte Objektbeschreibungen genutzt. Eine Objektbeschreibung beschreibt dabei grundlegende Eigenschaften eines Objekts, welche innerhalb solcher abstrakten Testmethoden benötigt werden. Beispiele für relevante Eigenschaften wären Klassennamen, Attributinformationen oder auch die Namen von Gettern und Settern. Durch diese Maßnahme kann in diesem Fall die Parameterliste auf eine Länge von zwei reduziert werden. Aufgrund der vorliegenden Objektbeschreibung ist beispielsweise die in Zeile fünf aufgerufene Methode in der Lage, ein Objekt zu instanziieren, sobald dieses implementiert wird. Falls noch keine Implementierung für dieses Objekt vorliegt, wird lediglich eine Fehlermeldung innerhalb des Tests geworfen, sodass das Projekt kompilieren kann und andere Tests ausgeführt werden können. Innerhalb der Test-Library wird eine Objektbeschreibung durch ein einfaches Objekt abgebildet (siehe Listing 2).

```
1  public class ObjectDescription {
2      private String className;
3      private String classPath;
4
5      private String getToOne;
6      private String getToMany;
7      private String setToOne;
8      private String setToMany;
9
10     private Attribute[] attributes = new Attribute[]{};
11
12     ...
13 }
```

Listing 2: Objektbeschreibung im Programmcode

Während der Testausführung kann der Inhalt dieses Objekts durch die Deserialisierung von mitgelieferten JSON-Dateien geladen werden. Listing 3 liefert ein Beispiel für eine solche JSON-Datei.

```
{
  "className": "DesktopPC",
  "classPath": "thkoeln.praktikum.DesktopPC",
  "getToOne": "getDesktopPC",
  "getToMany": "getDesktopPCs",
  "setToOne": "setDesktopPC",
  "setToMany": "setDesktopPCs",
  "attributes": [
    {
      "name": "storage",
      "type": "Integer",
      "serializedValue": "200"
    }, {
      "name": "energyConsumption",
      "type": "Integer",
      "serializedValue": "400"
    }
  ]
}
```

Listing 3: Objektbeschreibung im JSON Format

6.1.4 Prinzipien bei dem Test-Design

Automatische Tests sollen Lernenden schnell erstes Feedback generieren und repräsentieren außerdem bestimmte Anforderungen, welche an die Lösungen der Lernenden gestellt werden. Deshalb muss sich ein Lehrender beim Schreiben der Tests darüber im klaren sein, welche konkreten Anforderungen in Form von Tests veräußert werden sollen. Zugleich muss darauf geachtet werden, dass die Freiheit von Lernenden bei der Aufgabenbearbeitung nicht zu sehr durch gegebene Tests beschränkt wird. Laut Bente (2020, Interview, siehe Anhang 2.2.3, Z. 376-392) muss allerdings eine gewisse Eindeutigkeit der Lösung gegeben sein, damit automatisch getestet werden kann. Bente (2020, Interview, siehe Anhang 2.2.3, Z. 412-418) empfiehlt daher auf *Contract Testing* oder auch *Black Box Testing* zurückzugreifen, um die Freiheit bei der Aufgabenbearbeitung zu bewahren. Insofern müssen bestimmte Vorgaben gemacht werden. Hier gilt: So viel wie nötig, so wenig wie möglich.

Reitano (2021, Interview, siehe Anhang 2.2.4, Z. 276-296) merkt außerdem an, dass funktionale Aspekte einfach überprüfbar sind, weil diese klar definiert werden können. Architektonische Aspekte dahingegen sind schwierig überprüfbar, da es im Softwaredesign mehrere korrekte Lösungen gibt. Bei Tests, welche architektonische Aspekte abdecken

sollen, muss auf Seiten der Lehrenden genau überlegt werden, ob es sich im Einzelfall um Vorgaben oder Empfehlungen handelt. Bei Empfehlungen muss den Lernenden deutlich gemacht werden, dass es sich auch um solche handelt. Beispielsweise kann es auch vorkommen, dass das Implementieren einer bidirektionalen Beziehung eine valide Designentscheidung ist, obwohl in den meisten Fällen davon abgeraten wird. Laut Kohls (2020, Interview, siehe Anhang 2.2.2, Z. 406-419) hilft es außerdem im Vorfeld zu überlegen, was mögliche Fehlkonzepte sind und anhand deren zu testen, um besseres Feedback auf architektonischer Ebene geben zu können.

6.2 Aufgabenindividualisierung

Das bereits in der Übersicht erwähnte Divekit integriert optional nutzbare Funktionalitäten zur Aufgabenindividualisierung. Durch die automatische Aufgabenindividualisierung soll das einfache Kopieren von Lösungen anderer erschwert werden. Im Gegensatz zu den Individualisierungsmechanismen, welche der Praktomat (siehe Kapitel 4.1.4) bietet, stellt das Divekit dedizierte Individualisierungsmöglichkeiten bereit, welche auf die Anforderungen der praxisorientierten Informatiklehre abgestimmt sind. Außerdem müssen innerhalb der Dateien einer Aufgabe keine Kontrollstrukturen definiert werden, um das Erstellen einer individualisierbaren Aufgabe zu ermöglichen. Die Individualisierung wird durch Variablen beziehungsweise Platzhaltern innerhalb individualisierbarer Aufgaben ermöglicht. Das Prinzip ähnelt den parametrisierten Mathematikaufgaben, welche in Kapitel 4.1.1 thematisiert wurden. Aufgaben können mithilfe des Divekits auf drei verschiedene Arten variiert werden:

- *Objekt-Individualisierung*: Variiert die Eigenschaften einzelner Objekte (z.B. die Namen und Attribute einer Klasse)
- *Beziehungs-Individualisierung*: Bildet eine Menge an Beziehungen zufällig auf eine Menge an Objekt-Paaren ab
- *Logik-Individualisierung*: Ermöglicht das Definieren von Aufgabenteilen, welche jeweils einer Teilmenge von Lernenden zugeordnet werden

Schlussendlich kann durch das Kombinieren mehrerer Individualisierungsarten der Individualisierungsgrad gesteigert werden, sodass einzelne generierte Aufgaben mehr voneinander abweichen. Ein hoher Individualisierungsgrad ist allerdings nicht immer von Vorteil. Im Rahmen der Experteninterviews ergab sich, dass beispielsweise weniger individualisiert werden sollte, wenn Teamarbeit im Fokus steht. Dahingegen kann ein hoher Individualisierungsgrad nützlich sein, um das Kopieren von Lösungen zu verhindern (vgl. Interviewter

1 2020, Interview, siehe Anhang 2.2.1, Z. 264-270). Meist wird das Kopieren von Lösungen verhindert oder zumindest erschwert, indem immer wieder neue Aufgaben definiert werden, sodass alte Lösungen irrelevant werden. Folglich werden jedes mal Aufgaben verworfen, weil diese schon bekannt sind. Wenn allerdings mit individualisierten Aufgaben gearbeitet wird, können diese über Jahre weiterentwickelt werden und müssen nicht jedes Mal ersetzt werden (vgl. Reitano 2021, Interview, siehe Anhang 2.2.4, Z. 260-270).

Auch wenn ein hoher Individualisierungsgrad angestrebt wird, so sollte die Fachlichkeit der Aufgabe nicht darunter leiden, wenn diese für den jeweiligen Kurs relevant ist. Deshalb ist die Fachlichkeit einer Aufgabe höher zu bewerten als der Individualisierungsgrad, wenn innerhalb eines Kurses der Schritt von der Fachlichkeit zur Technik hin relevant ist (vgl. Bente 2020, Interview, siehe Anhang 2.2.3, Z. 303-312). Dies gilt infolgedessen nicht für Programmierkurse, bei welchen ausschließlich technische Implementierungen den Schwerpunkt bilden.

6.2.1 Objekt-Individualisierung

Die Objekt-Individualisierung erlaubt das Definieren von mehreren Varianten eines Objekts, wobei während der Individualisierung jeweils nur eine Teilmenge der Varianten zufällig ausgewählt wird. Diese Teilmenge umfasst in den meisten Fällen jedoch nur eine einzelne Variante, sodass aus Gründen der Einfachheit im Folgenden nur dieser Fall betrachtet wird. Innerhalb einer Variante können eine Menge von Eigenschaften konfiguriert werden, welche das Objekt definieren sollen. Pflicht ist hierbei nur das Definieren der Eigenschaft *id*. Sämtliche zusätzlich konfigurierte Eigenschaften können zudem mit einer beliebigen Tiefe an Verschachtelungen festgelegt werden. Dies wird anhand eines Beispiels deutlich. In Listing 4 wird ein Objekt *Computer* definiert. Diesem werden die zwei Varianten *DesktopPC* und *Laptop* untergeordnet, welche sich in der Benennung und den dazugehörigen Attributen unterscheiden. Während des Individualisierungsvorgangs wird für einen Lernenden nun eine der beiden Varianten zufällig ausgewählt. Bei diesem Vorgang wird außerdem gespeichert, welcher Lernender welche Varianten gewürfelt hat. So können diesem bei verschiedenen aufeinander aufbauenden Aufgaben immer die gleichen Objektvarianten zugeordnet werden.

```
1 {
2     "ids": "Computer",
3     "objectVariations": [
4         {
5             "id": "DesktopPC",
6             "De": "DesktopPC",
```

```
            "Attr1": {
                "": "storage",
                "Type": "Integer",
                "Value": "200",
                "De": "Speicher"
            },
            "Attr2": {
                "": "energyConsumption",
                "Type": "Integer",
                "Value": "400",
                "De": "Energieverbrauch"
            }
        },
        {
            "id": "Laptop",
            "De": "Laptop",
            "Attr1": {
                "": "batteryLife",
                "Type": "Integer",
                "Value": "8",
                "De": "Batterielaufzeit"
            },
            "Attr2": {
                "": "keyboardLayout",
                "Type": "String",
                "Value": "DEU",
                "De": "Tastaturlayout"
            }
        }
    ],
    "variableExtensions": [ "Basic", "Plural", "Getter", "Setter" ]
}
```

Listing 4: Objekt-Individualisierung im JSON Format

Anschließend werden aus den definierten Eigenschaften einer Variante dazugehörige Variablen generiert, welche bei der Aufgabenerstellung genutzt werden können, um eine Individualisierung zu ermöglichen. Unter der Annahme, dass ein Lernender die Objektvariante *DesktopPC* gewürfelt hat, können in Listing 5 die daraus generierten Variablen eingesehen werden. Nach genauerer Betrachtung der generierten Variablen fällt auf, dass Variablen vorhanden sind, welche nicht innerhalb der Objektvarianten definiert wurden. Beispielsweise wurden keine Variablen mit den Namen *Class* oder *GetToOne* konfiguriert. Dennoch sollten diese generiert werden, weil es sich dabei um Variablen handelt, welche bei der späteren Aufgabenerstellung benötigt werden. Bei genannten Variablen handelt

es sich um solche, welche einfach von bestehenden Variablen abgeleitet werden können. Der Name eines Getters fügt beispielsweise nur das Wort *get* vorne an die vorhandene Variable *id* an. Manche Variablen, welche aus Bestehenden generiert werden, ändern sogar nur ihren Namen, nicht aber ihren Wert. So wurde zum Beispiel aus der Variable *id* die Variable *Class* generiert, obwohl sich der Wert *DesktopPC* nicht geändert hat. Dies ist dadurch begründet, dass Variablennamen fachlicher Natur sein sollten, weil der Name einer Variable ausdrückt, welchem Zweck die Variable dient. Erweiterungen dieser Art werden in einer zusätzlichen Konfigurationsdatei hinterlegt. Dort wird jeweils eine bereits bestehende Variable genutzt, um vor und nach dieser fest definierte Wörter anzuhängen.

```
Computer: {
    Computer: "DesktopPC",
    Computers: "DesktopPCs",
    ComputerDe: "DesktopPC",
    ComputerClass: "DesktopPC",
    ComputerClassPath: "thkoeln.praktikum.DesktopPC",
    Computer_Attr1: "storage",
    Computer_Attr1Type: "Integer",
    Computer_Attr1Value: "200",
    Computer_Attr1De: "Speicher",
    Computer_Attr2: "energyConsumption",
    Computer_Attr2Type: "Integer",
    Computer_Attr2Value: "400",
    Computer_Attr2De: "Energieverbrauch",
    ComputerGetToOne: "getDesktopPC",
    ComputerGetToMany: "getDesktopPCs",
    ComputerSetToOne: "setDesktopPC",
    ComputerSetToMany: "setDesktopPCs"
}
```

Listing 5: Generierte Variablen der Objekt-Individualisierung

6.2.2 Beziehungs-Individualisierung

Die Beziehungs-Individualisierung knüpft an der Objekt-Individualisierung an. Auf Basis vorher definierter Objekte werden Beziehungen zwischen einzelnen Objekten festgelegt, damit später die Multiplizitäten gewürfelt werden können. Diese werden allerdings nicht wahllos verwürfelt, weil ansonsten die Fachlichkeit darunter leiden würde. Außerdem wäre es nicht gerecht, wenn ein Lernender beispielsweise nur *Many to Many* -Beziehungen und ein anderer nur *One to One* -Beziehungen implementieren müsste. Daher wird von Anfang an festlegt, wie viele von welchen Beziehungstypen es insgesamt geben wird. Anschließend werden diese auf konfigurierte Paare von Objekten abgebildet. Folglich ist bei

der Aufgabenerstellung darauf zu achten, dass jeder definierte Beziehungstyp fachlich in Bezug auf jedes definierte Paar an Objekten begründet werden kann. Das Festlegen von einer Menge an Beziehungstypen und dazugehörigen Paaren von Objekten kann dabei in mehreren unabhängigen Abschnitten passieren. Listing 6 stellt einen solchen Abschnitt dar, welcher zwei Beziehungstypen und zwei Paare von Objekten definiert. Das Beispiel geht hierbei davon aus, dass vorab neben dem Objekt *Computer* die Objekte *User* und *Periphery* definiert wurden.

Auch wenn durch die Kombination der Objekt- und Beziehungs-Individualisierung der Individualisierungsgrad stark gesteigert werden kann, ist zu beachten, dass das Erstellen von Aufgaben mit komplexeren fachlichen Zusammenhängen in diesem Fall stark erschwert wird. Das Konstruieren einer Geschichte für den Aufgabenkontext wird zunehmend schwerer, wenn neben den Objektvarianten zusätzlich die Beziehungen zwischen diesen variieren können. Aus diesem Grund eignet sich die Beziehungs-Individualisierung eher für Programmieraufgaben, bei denen fachliche Zusammenhänge weniger relevant sind.

```
{
    "relationShips": [
        {
            "id": "Rel1",
            "relationType": "OneToMany"
        },
        {
            "id": "Rel2",
            "relationType": "ManyToMany"
        }
    ],
    "relationObjects": [
        {
            "id": "RelComputerUser",
            "obj1": "Computer",
            "obj2": "User"
        },
        {
            "id": "RelComputerPeriphery",
            "obj1": "Computer",
            "obj2": "Periphery"
        }
    ]
}
```

Listing 6: Beziehungs-Individualisierung im JSON Format

Auch aus den konfigurierten möglichen Beziehungen werden nach der Verwürfelung Variablen generiert. Hierbei kann zwischen drei grundlegenden Arten von Variablen unterschieden werden. Dies umfasst Variablen, welche dem Beziehungstyp selbst zugeordnet werden, Variablen, welche zu dem ersten Objekt der Beziehung gehören und Variablen, welche zu dem zweiten Objekt der Beziehung gehören. Variablen vom ersten Typ werden aus einer zusätzlichen Konfigurationsdatei geladen, welche jedem Beziehungstyp einige Eigenschaften zuweist. So kann beispielsweise einer *One to Many* -Beziehung die Notation *1-n* zugeordnet werden. Variablen vom zweiten und dritten Typ ergeben sich aus den bereits vorhandenen Variablen, welche den einzelnen Objekten der Beziehung zugewiesen wurden. Listing 7 stellt einen Auszug an generierten Variablen für die Beziehungen dar.

Die auf Basis der definierten Beziehungen generierten Variablen werden außerdem aus zwei Perspektiven gepflegt. Eine Perspektive umfasst Variablen, welche darüber Auskunft geben, welche Objekte einem Beziehungstyp zugeordnet wurden (siehe Block *Rel1* und Block *Rel2*). Die zweite Perspektive beschreibt die Gegenrichtung, denn diese umfasst Variablen, welche einem Paar an Objekten jeweils einen Beziehungstyp zuweisen (siehe Block *RelComputerPeriphery* und Block *RelComputerUser*). Die erste Perspektive ist vor allem bei automatischen Tests innerhalb von individualisierten Aufgaben relevant. Durch diese können einem vorab geschriebenen Test, welcher beispielsweise die korrekte Implementierung einer *One to Many* -Beziehung testet, zwei Objekte zugewiesen werden, für welche der Beziehungstyp implementiert werden soll. Dies würde über die Variablen *Rel1_Obj1* und *Rel1_Obj2* passieren. Die zweite Perspektive ist mehr aus Sicht der Aufgabenstellung relevant. Mithilfe der dazugehörigen Variablen kann beschrieben werden, welcher Beziehungstyp zwischen zwei Objekten implementiert werden soll. Hier sind vor allem Variablen wie zum Beispiel *RelComputerPeriphery_Description* relevant.

```
{
    Rel1: {
        Rel1_Description: "one to many",
        Rel1_Obj1: "DesktopPC",
        Rel1_Obj2: "Screen",

        ...
    },
    RelComputerPeriphery: {
        RelComputerPeriphery_Description: "one to many",
        RelComputerPeriphery_Obj1: "DesktopPC",
        RelComputerPeriphery_Obj2: "Screen",

        ...
```

```
    },
    Rel2: {
        Rel2_Description: "many to many",
        Rel2_Obj1: "DesktopPC",
        Rel2_Obj2: "Developer",

        ...
    },
    RelComputerUser: {
        RelComputerUser_Description: "many to many",
        RelComputerUser_Obj1: "DesktopPC",
        RelComputerUser_Obj2: "Developer",

        ...
    }
}
```

Listing 7: Generierte Variablen der Beziehungs-Individualisierung

6.2.3 Logik-Individualisierung

Die Variablengenerierung der Logik-Individualisierung ähnelt der Variablengenerierung der Objekt-Individualisierung. Dabei unterscheidet sich die Logik-Individualisierung allerdings von den anderen Individualisierungsarten darin, dass generierte Variablen nicht zentraler Bestandteil der Individualisierung sind. Wichtiger ist hier, dass eine Variante überhaupt zufällig ausgewählt wird. Wie Listing 8 verdeutlicht, kann in Einzelfällen sogar auf das Definieren sämtlicher Eigenschaften verzichten werden. Die grundlegende Idee hinter der Logik-Individualisierung ist, dass gewürfelten Varianten entscheiden, ob eine jeweilige Datei in das Aufgabenprojekt eines Lernenden kopiert wird oder nicht. Entschieden wird dies anhand des Dateinamens. Wenn eine Datei einen Unterstrich gefolgt von einer gewürfelten *id* enthält, so wird diese Datei in das Aufgabenprojekt des Lernenden übernommen, anderenfalls nicht. Wenn von dem Beispiel in Listing 8 ausgegangen wird, so würden entweder alle Dateien mit einem "*_Datamodel*" im Namen oder alle Dateien mit einem "*_UseCaseDiagram*" im Namen übernommen werden.

Dieser Mechanismus ermöglicht das Definieren von grundlegend verschiedenen Aufgaben, wobei ein Lernender nur jeweils eine dieser Aufgaben lösen muss. Dies ähnelt dem Prinzip der Fragenpools, welche auch innerhalb Ilias (siehe Kapitel 4.1.2) Anwendung finden. Infolgedessen muss auch hier bei der Aufgabenerstellung darauf geachtet werden, dass der Schwierigkeitsgrad einzelner Aufgabenvarianten nicht zu sehr voneinander abweicht. Anzumerken ist jedoch, dass auch mehrere kleinere Aufgabensammlungen definiert wer-

den können, sodass eine Aufgabe pro Aufgabensammlung übernommen und nicht aus der Gesamtheit an verfügbaren Aufgaben ausgewählt wird. Schlussendlich kann dieser Mechanismus auch innerhalb einer Aufgabe genutzt werden, um Lernende jeweils andere fachliche Anforderungen umsetzen zu lassen oder um beispielsweise das Layout von Diagrammen, welche Teil der Aufgabenstellung sind, zu variieren.

```
{
    "id": "Exercise1",
    "logicVariations": [
        {
            "id": "Datamodel"
        },
        {
            "id": "UseCaseDigram"
        }
    ]
}
```

Listing 8: Logik-Individualisierung im JSON Format

6.2.4 Aufgabenindividualisierung mittels generierter Variablen

Nachdem auf Basis der drei Individualisierungsarten dementsprechende Variablen generiert wurden, stehen diese bei der Aufgabenerstellung zur Verfügung. In diesem Kapitel wird demonstriert, wie Variablen in den verschiedenen Teilen einer Aufgabe genutzt werden können, um diese zu individualisieren. Besagte Variablen ergeben sich dabei aus der Objekt- und Beziehungs-Individualisierung. Dennoch findet auch hier eine Logik-Individualisierung statt. Sämtliche noch nicht individualisierte Beispieldateien innerhalb dieses Kapitels enthalten im Namen ein *_Datamodel*. Infolgedessen werden diese Dateien gruppiert und einer Aufgabe zugeordnet. In diesem Fall müsste die Aufgabe *Datamodel* gelöst werden, weil diese gewürfelt wurde. Wenn diese nicht gewürfelt worden wäre, müsste die andere in Listing 8 konfigurierte Aufgabe gelöst werden.

Aus diesem Grund beziehen sich die folgenden Beispiele auf den gleichen Aufgabenkontext, welcher bereits in den vorherigen Kapiteln thematisiert wurde. Allerdings wurde neben den Objekten *Computer*, *User* und *Periphery* noch das Objekt *Benchmark* hinzugefügt, welches von dem *Computer*-Objekt referenziert werden soll.

Individualisierung der Aufgabenstellung

Da Aufgaben in Form eines Git-Projekts zur Verfügung gestellt werden, bietet es sich

an, die Aufgabenstellung als Markdown-Datei bereitzustellen. Innerhalb der Aufgabenstellung können Variablen platziert werden, damit der Aufgabentext bei einer späteren Individualisierung an die jeweiligen gewürfelten Varianten angepasst wird. Ein Beispiel hierfür ist Listing 9 zu entnehmen.

```
Implementieren Sie das folgende Geflecht aus Klassen mithilfe von Spring
Data JPA:

Die Klasse $Benchmark$ (auf Deutsch: $BenchmarkDe$) speichert die
Ergebnisse eines einzelnen Benchmarks. Dabei wird jeder $Benchmark$
genau einem $Computer$ zugeordnet und von diesem referenziert. Zudem
soll zwischen $Computer$ und $Periphery$ (auf Deutsch: $PeripheryDe$)
eine $RelComputerPeriphery_Description$ -Beziehung und zwischen
$Computer$ und $User$ (auf Deutsch: $UserDe$) eine
$RelComputerUser_Description$ -Beziehung implementiert werden. Folglich
referenziert ein $Computer$ eine Liste von $PeripheriesDeDativ$ und eine
Liste von $UsersDeDativ$.
```

Listing 9: Aufgabenstellung: Exercise1_Datamodel.md

Meistens haben genutzte Variablen ihren Ursprung in der Objekt-Individualisierung. Einzig die Eigenschaft *Description* der Variablen *RelComputerPeriphery* und *RelComputerUser*, welche der Beziehungs-Individualisierung angehören, wird genutzt, um den jeweiligen gewürfelten Beziehungstyp anzugeben. Auch wenn die obigen Formulierungen eher technischer Natur sind, können auf diese Weise auch ausführliche fachlich formulierte Aufgabenstellungen verfasst werden. Jedoch ist hierbei anzumerken, dass die Komplexität der Grammatik, welche die jeweils genutzte Sprache fordert, das Erstellen solcher Aufgabentexte beeinflusst. Für den obigen Aufgabentext mussten beispielsweise die Variablen *PeripheriesDeDativ* und *UsersDeDativ* definiert werden, um einen Satz zu konstruieren, welcher der deutschen Grammatik gerecht wird. Infolgedessen kann die Wahl der Sprache die Aufgabenerstellung erschweren. Hier könnten sprachbezogene Mechanismen Abhilfe schaffen, welche die Aufgabenstellung in dieser Hinsicht erleichtern. Schlussendlich würde nach der Individualisierung ein Aufgabentext resultieren, wie er in Listing 10 dargestellt wird.

```
Implementieren Sie das folgende Geflecht aus Klassen mithilfe von Spring
Data JPA:

Die Klasse ProcessorBenchmark (auf Deutsch: Prozessor-Benchmark)
speichert die Ergebnisse eines einzelnen Benchmarks. Dabei wird jeder
ProcessorBenchmark genau einem DesktopPC zugeordnet und von diesem
referenziert. Zudem soll zwischen DesktopPC und Screen (auf Deutsch:
Bildschirm) eine One to Many -Beziehung und zwischen DesktopPC und
Developer (auf Deutsch: Entwickler) eine Many to Many -Beziehung
implementiert werden. Folglich referenziert ein DesktopPC eine Liste
von Bildschirmen und eine Liste von Entwicklern.
```

Listing 10: Individualisierte Aufgabenstellung: Exercise1.md

Individualisierung von Diagrammen

Das Aufgabenbeispiel soll weitergeführt werden und zusätzlich ein Datenmodell in Form eines Diagramms entsprechend individualisiert werden. Abbildung 5 stellt ein Diagramm dar, welches Variablen enthält, um das Datenmodell an die gewürfelte Variante anzugleichen. Auch hier entstammen sämtliche Variablen der Objekt-Individualisierung. Lediglich die Eigenschaft *Umlet* der Variablen *RelComputerUser* und *RelComputerPeriphery* wird genutzt, um je nach gewürfeltem Beziehungstyp die jeweiligen Notationen in das Diagramm einzutragen. Abbildung 6 dahingegen zeigt die individualisierte Variante an, welche sich nach dem Ersetzen der Variablen ergeben würde. In diesem Beispiel sind die dargestellten Diagramme teil der Aufgabenstellung, dennoch können auf die gleiche Weise auch Musterlösungen in Form von Diagrammen bereitgestellt werden.

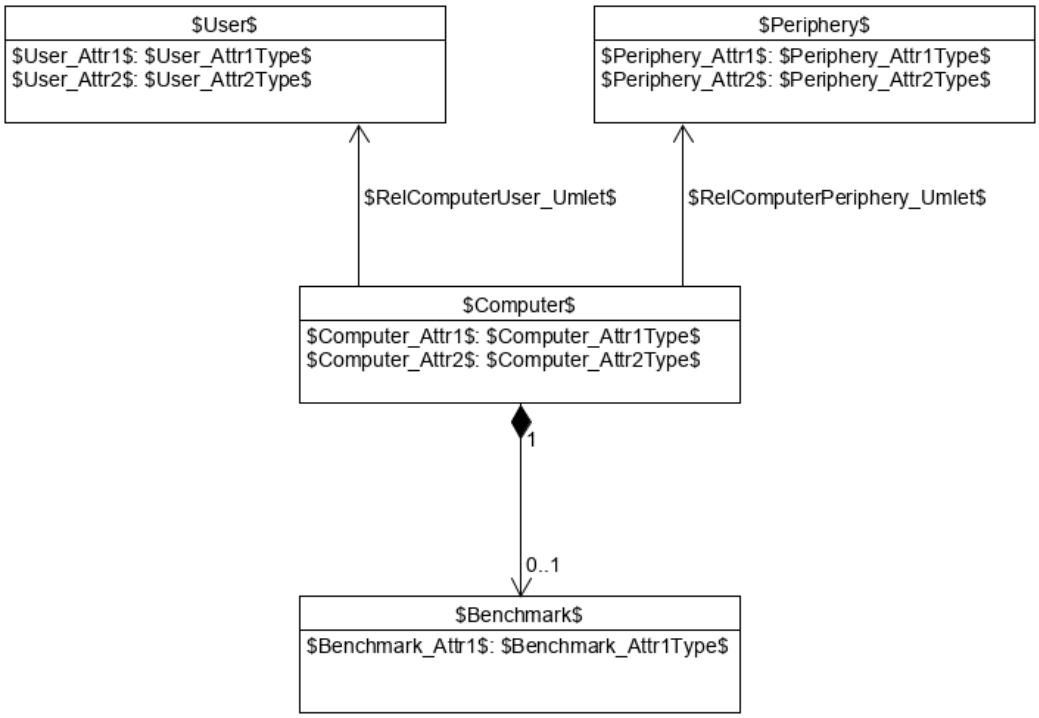

Abbildung 5: Diagramm: Datenmodell_Datamodel.jpg

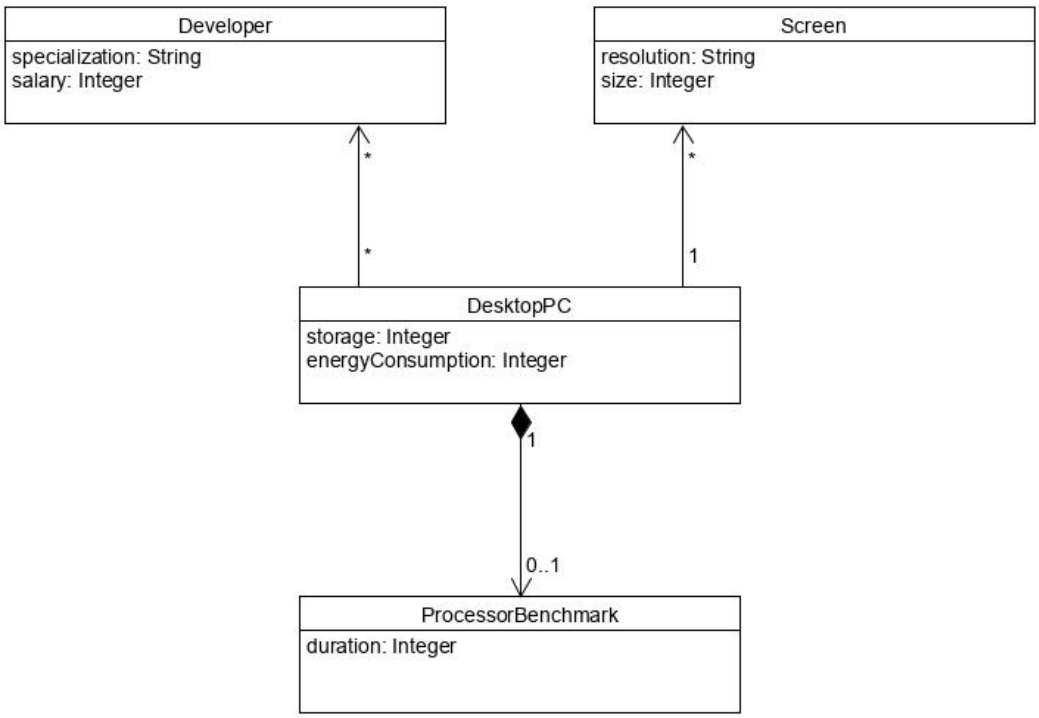

Abbildung 6: Diagramm: Individualisiertes Datenmodell.jpg

Individualisierung von automatischen Tests

Wenn die Aufgabenstellung variiert, muss natürlich von jedem Lernenden jeweils eine etwas andere Lösung implementiert werden. Um diese automatisch überprüfen zu können, müssen auch die automatischen Tests dementsprechend angepasst werden. Dazu könnten Variablen innerhalb von Test-Programmcode genutzt werden. Allerdings reicht es häufig aus, wenn nur die Objektbeschreibungen (siehe Kapitel 6.1.3), auf welchen Tests basieren, individualisiert werden. Dies gilt jedoch nur bei Tests mit geringer Komplexität, weil Objektbeschreibungen nur grundlegende Informationen über Objekte bereitstellen. Wenn Tests also Informationen benötigen, welche über die Objektbeschreibungen hinaus gehen, müssen diese Informationen anderweitig bereitgestellt und gegebenenfalls mithilfe von Variablen individualisiert werden. Ein Beispiel für eine Objektbeschreibung auf Basis von Variablen kann Listing 11 entnommen werden.

```
{
  "className": "$Computer$",
  "classPath": "$ComputerClassPath$",
  "getToOne": "$ComputerGetToOne$",
  "getToMany": "$ComputerGetToMany$",
  "setToOne": "$ComputerSetToOne$",
  "setToMany": "$ComputerSetToMany$",
  "attributes": [
    {
      "name": "$Computer_Attr1$",
      "type": "$Computer_Attr1Type$",
      "serializedValue": "$Computer_Attr1Value$"
    }, {
      "name": "$Computer_Attr2$",
      "type": "$Computer_Attr2Type$",
      "serializedValue": "$Computer_Attr2Value$"
    }
  ]
}
```

Listing 11: Objektbeschreibung: $Computer$_Datamodel.java

In Listing 12 wird das Beispiel von automatischen Tests aufgegriffen, welche überprüfen, ob eine Beziehung zwischen zwei Objekten korrekt implementiert wurde. Dabei werden drei verschiedene Tests aufgerufen, welche einen jeweils anderen Beziehungstyp prüfen. Über die mitgelieferten Objektbeschreibungen werden die Tests hierbei parametrisiert. Da Objektbeschreibungen nun individualisiert vorliegen, muss auch hier mit Variablen gearbeitet werden, um die jeweils gewürfelten Objekte passend anzugeben. Die ersten beiden Tests prüfen Beziehungen, welche verwürfelt werden können. Deshalb finden dort

Variablen Anwendung, welche der Beziehungs-Individualisierung entstammen. Sogar der Name der Tests kann entsprechend mithilfe der Variablen *Rel1_Obj1* und *Rel1_Obj2* angepasst werden. Der dritte Test überprüft die Beziehung zwischen den Objekten *Computer* und *Benchmark*. Da diese Beziehung nicht verwürfelt wird, genügen hier Variablen aus der Objekt-Individualisierung. Listing 13 stellt die individualisierte Variante der Tests dar, welche später den Lernenden vorliegt.

```
@Test
public void oneToMany$Rel1_Obj1$$Rel1_Obj2$Test() throws Exception {
    genericRelationsTests.oneToManyTest(
        getObjectDescription("$Rel1_Obj1$"),
        getObjectDescription("$Rel1_Obj2$"));
}

@Test
public void manyToMany$Rel2_Obj1$$Rel2_Obj2$Test() throws Exception {
    genericRelationsTests.manyToManyTest(
        getObjectDescription("$Rel2_Obj1$"),
        getObjectDescription("$Rel2_Obj2$"));
}

@Test
public void oneToOne$Computer$$Benchmark$Test() throws Exception {
    genericRelationsTests.oneToOneVOTest(
        getObjectDescription("$Computer$"),
        getObjectDescription("$Benchmark$"));
}
```

Listing 12: Tests: RelationTests_Datamodel.java

```
@Test
public void oneToManyDesktopPCScreenTest() throws Exception {
    genericRelationsTests.oneToManyTest(
        getObjectDescription("DesktopPC"),
        getObjectDescription("Screen"));
}

@Test
public void manyToManyDesktopPCDeveloperTest() throws Exception {
    genericRelationsTests.manyToManyTest(
        getObjectDescription("DesktopPC"),
        getObjectDescription("Developer"));
}

```

```
15 @Test
16 public void oneToOneDesktopPCProcessorBenchmarkTest() throws Exception {
17     genericRelationsTests.oneToOneVOTest(
18         getObjectDescription("DesktopPC"),
19         getObjectDescription("ProcessorBenchmark"));
20 }
```

Listing 13: Individualisierte Tests: RelationTests.java

6.3 Aufgabenbereitstellung

Nachdem Aufgaben mithilfe des Divekits individualisiert wurden, stehen diese den Lernenden allerdings noch nicht zur Verfügung. Bevor jedem Lernenden ein Aufgabenprojekt bereitgestellt wird, nehmen bestimmte Automatismen letzte Änderungen an den Aufgaben vor. Sobald dies geschehen ist, werden die Aufgabenprojekte verteilt und auf Gitlab zur Verfügung gestellt.

6.3.1 Modifikation von Aufgabenbestandteilen

Wie oben beschrieben werden vor der eigentlichen Aufgabenbereitstellung einige Automatismen ausgeführt, um letzte Änderungen auf die Aufgabenprojekte anzuwenden. Im Folgenden werden die drei bisher implementierten Automatismen aufgezählt und erläutert.

Automatisches Löschen von Lösungen
Das Origin-Projekt bildet die Basis für das Code- und Test-Projekt. Oft enthält das Origin-Projekt Musterlösungen für Aufgaben, welche während der Aufgabenstellung von Lehrenden geschrieben werden. Diese können automatisch rausgelöscht werden, indem vorab Markierungen wie beispielsweise *deleteParagraph* oder *replaceParagraph* um Lösungsbestandteile platziert werden. Hierbei muss allerdings angemerkt werden, dass nur später generierte Aufgaben von diesen Änderungen betroffen sind und dass das Origin-Projekt unangetastet bleibt.

Zuordnung von Dateien zum Code- und Test-Projekt
Bereits in Kapitel 6 wird erwähnt, dass es sich bei dem Code- und Test-Projekt um zwei unterschiedliche Projekte handelt, welche jeweils andere Dateien enthalten können. Lernende haben dabei nur Zugriff auf das Code-Projekt und können deshalb nicht auf Dateien innerhalb des Test-Projekts zugreifen. Damit Dateien des Origin-Projekts nun bei der Aufgabenbereitstellung in den jeweils richtigen Projekten landen, muss bei der Aufgabenerstellung die Möglichkeit geschaffen werden, dies als Lehrender zu definieren.

Der Automatismus dahinter funktioniert ähnlich wie die Logik-Individualisierung (siehe Kapitel 6.2.3). Je nach gewähltem Dateinamen entscheidet sich, ob eine Datei in die jeweiligen Projekte kopiert wird:

- Enthält der Dateiname ein *_coderepo*, wird die Datei nur in das Code-Projekt kopiert.
- Enthält der Dateiname ein *_testrepo*, wird die Datei nur in das Test-Projekt kopiert.
- Enthält der Dateiname ein *_norepo*, wird die Datei weder in das Code-Projekt noch in das Test-Projekt kopiert.
- Trifft keine der obigen Bedingungen zu, wird die Datei in beide Projekte kopiert.

Aufgabenbestandteile erweitern

Ein Großteil der Aufgaben wird mithilfe von Variablen individualisiert. Dennoch sollten Variablen in bestimmten Dateitypen wie beispielsweise Bildern nicht enthalten sein, da dementsprechende Dateien ansonsten aufgrund des veränderten Inhalts unlesbar gemacht werden. Trotzdem ist in Kapitel 6.2.4 ein Beispieldiagramm enthalten, welches individualisiert wird. Dazu wird das Diagramm im Rohformat (z.B. XML) mit Variablen versehen, sodass die Individualisierung innerhalb des Rohformats stattfinden kann. Das Divekit ermöglicht nun das Definieren von Dateimodifikatoren, welche auf Basis bestehender Dateien neue Dateien innerhalb des Projekts generieren. Folglich kann dieser Mechanismus dazu genutzt werden, um Diagramme, welche in einem Rohformat vorliegen, in ein Bildformat zu überführen. Somit können Diagramme Lernenden immer im Bildformat bereitgestellt werden, sodass diese die Diagramme nicht mithilfe einer bestimmten Software öffnen müssen.

6.3.2 Aufgabenverteilung

Wenn mit dem Divekit optional eine Individualisierung stattgefunden hat und die Code- und Test-Projekte innerhalb des Divekits vorliegen, kann die Verteilung der Aufgaben beginnen. Dies geschieht über die Gitlab API. Parallel wird jeweils eine konfigurierte Anzahl an Projekten angelegt. Zuerst werden leere Code- und Test-Projekte erzeugt, damit anschließend die nötigen Dateien in die Projekte hochgeladen werden können. Wie in Kapitel 6.1.1 beschrieben, muss das Code-Projekt mit dem Test-Projekt verlinkt sein, um gewünschte Funktionalitäten bereitstellen zu können. Nachdem jeweils ein Code- und Test-Projekt mit dazugehörigem Inhalt in Gitlab hochgeladen wurde, verlinkt das Divekit die beiden Projekte unter Benutzung von Umgebungsvariablen miteinander. Nach Abschluss dieses Schrittes liegt das jeweilige Code- und Test-Projekt in fertiger Form in

Gitlab vor. Nun werden dem Lernenden, für welchen das Code-Projekt vorgesehen ist, die nötigen Zugriffsrechte automatisch zugewiesen. Wenn beschriebener Automatisierungsvorgang abgeschlossen ist, wird eine dementsprechende Übersichtsseite generiert. Diese ordnet jedem Nutzernamen eines Lernenden nicht nur das jeweilige Code- und Test-Projekt, sondern auch die dazugehörige Testseite zu. Deshalb wird die Übersichtsseite von Lehrenden genutzt, um während eines Praktikums eine Übersicht über alle Lernenden und deren Aufgabenprojekte zu gewähren. In Abbildung 7 wird eine solche Übersichtsseite beispielhaft dargestellt.

Group	Code Repo	Test Repo	Test Page
jintveen	https://git.st.archi-lab.io/students/st1/ws20/m3/st1-praktikum_group_c9ce4a36-2cae-4cc6-bf89-6e665dc13992	https://git.st.archi-lab.io/students/st1/ws20/m3/tests/st1-praktikum_tests_group_c9ce4a36-2cae-4cc6-bf89-6e665dc13992	http://students.pages.st.archi-lab.io/st1/ws20/m3/tests/st1-praktikum_tests_group_c9ce4a36-2cae-4cc6-bf89-6e665dc13992
hpeter	https://git.st.archi-lab.io/students/st1/ws20/m3/st1-praktikum_group_55e747d5-f699-4d2d-8cc7-a8787657798f	https://git.st.archi-lab.io/students/st1/ws20/m3/tests/st1-praktikum_tests_group_55e747d5-f699-4d2d-8cc7-a8787657798f	http://students.pages.st.archi-lab.io/st1/ws20/m3/tests/st1-praktikum_tests_group_55e747d5-f699-4d2d-8cc7-a8787657798f
fmüller	https://git.st.archi-lab.io/students/st1/ws20/m3/st1-praktikum_group_cc9b43d0-d676-419f-86e6-9c1c0e784709	https://git.st.archi-lab.io/students/st1/ws20/m3/tests/st1-praktikum_tests_group_cc9b43d0-d676-419f-86e6-9c1c0e784709	http://students.pages.st.archi-lab.io/st1/ws20/m3/tests/st1-praktikum_tests_group_cc9b43d0-d676-419f-86e6-9c1c0e784709
balbrecht	https://git.st.archi-lab.io/students/st1/ws20/m3/st1-praktikum_group_80d41979-ea1e-4ddb-8b0e-81affeca4de0	https://git.st.archi-lab.io/students/st1/ws20/m3/tests/st1-praktikum_tests_group_80d41979-ea1e-4ddb-8b0e-81affeca4de0	http://students.pages.st.archi-lab.io/st1/ws20/m3/tests/st1-praktikum_tests_group_80d41979-ea1e-4ddb-8b0e-81affeca4de0

Abbildung 7: Beispielhafte Übersichtsseite

7 Fallstudien

Einige der in den Kapiteln 4 und 6 vorgestellten Werkzeuge wurden in Kursen, in welchen Programmierung eine zentrale Rolle spielt, erprobt. Diese Kurse können als Fallstudien betrachtet werden, welche Erkenntnisse in Bezug auf diese Werkzeuge, aber auch auf thematisierte Konzepte aus der Programmierung (siehe Kapitel 2.2) liefern. Zusätzlich können Beobachtungen, welche in den Kursen gemacht wurden, auf diverse Konzepte aus der Didaktik (siehe Kapitel 2.1) zurückgeführt werden. Schlussendlich ist anzumerken, dass der Autor dieser Arbeit bei allen drei Fallstudien einer der Betreuer der Lernenden war.

7.1 Coding Essentials 1

7.1.1 Beschreibung des Falls

Der Kurs *Coding Essentials 1* ist ein Programmierkurs für Anfänger, welcher am Anfang des Studiengangs *Code & Context* eingegliedert ist (vgl. *Kurs Coding Essentials 1 (CS12)* o.D.). Der Kurs fand das erste mal im Wintersemester 2019 statt. An dem Kurs haben etwas über 20 Lernende teilgenommen, welche von drei Lehrenden betreut wurden. Das Lernziel von Coding Essentials 1 war es, dass Lernende Programmcode in der Programmiersprache *Java* schreiben können, essentielle Datenstrukturen kennenlernen und einfache algorithmische Vorgänge in Programmcode abbilden können. Die Basis bildeten eine Reihe von Aufgaben, welche nacheinander die relevanten Themen behandelten.

Verteilt wurden die Aufgaben über Gitlab (siehe Kapitel 4.1.3), wo jeder Lernender sein eigenes Projekt mitsamt den Aufgaben hatte. Das Erzeugen der Projekte für die Lernenden geschah über ein Programm, welches die Gitlab API nutzte. Wenn ein Lernender einen Commit auf sein Projekt absetzte, so wurden mithilfe einer in Gitlab definierten Pipeline JUnit-Tests ausgeführt, dessen Code einsehbar war. Die Testergebnisse wurden dann mithilfe eines Testseitengenerators in eine übersichtliche statische Seite überführt, auf welcher sowohl Lernende als auch Lehrende die Ergebnisse von den Tests einsehen konnten. Wichtig zu erwähnen ist dabei, dass das Erfüllen der Tests nicht Voraussetzung für ein Bestehen war. Das Bestehen des Kurses und eine damit verbundene Note war ausschließlich von einer letzten Programmieraufgabe abhängig.

Die Aufgaben wurden hauptsächlich in den Präsenzterminen bearbeitet, welche in einem Block stattgefunden haben. Die Präsenztermine wurden außerdem nach dem didaktischen Konzept Active Learning (siehe Kapitel 2.1.1) gestaltet. Infolgedessen wurden häufig kurze Inhaltsimpulse gegeben, anschließend darauf abgestimmte Aufgaben bearbeitet und ab-

schließend wurden jeweils aufgetretene Problemstellungen diskutiert. Während Lernende Aufgaben gemeinsam oder in Teamarbeit bearbeitet haben, sind Betreuer herum gegangen und haben Fragen beantwortet oder bei Problemen geholfen.

7.1.2 Erkenntnisse

Im Verlauf dieses Kurses konnte bei vielen Lernenden ein erheblicher Lernerfolg vermerkt werden. Die meisten Lernenden schienen intrinsisch motiviert (siehe Kapitel 2.1.2) zu sein, sodass zusätzliche extrinsische Motivation kaum nötig war. Dies könnte unter anderem dadurch begründet sein, dass ein motivierendes Lernklima vorhanden war. Betreuer begegneten den Lernenden auf Augenhöhe, was dazu führte, dass die Distanz von Lehrenden zu Lernenden sehr gering war. Sowohl Lehrende als auch Lernende hatten Spaß bei dem Bearbeiten der Aufgaben. Zusätzlich wird die Abwesenheit von externen Regulierungen wie Belohnungen oder Bestrafungen auch dazu beigetragen haben, dass bereits vorhandene intrinsische Motivation nicht negativ beeinflusst wurde.

Da Abläufe innerhalb des Kurses nach dem didaktischen Konzept Active-Learning gestaltet waren und eine hohe Anzahl an Betreuern verfügbar war, konnte stets schnell und ausführlich Feedback gegeben werden. Die vielen verfügbaren Ressourcen sind dadurch begründet, dass es sich hierbei um ein Pilotprojekt eines neuen Studiengangs handelte. Teilweise konnte auch Zeit dafür genutzt werden, Lernende darauf hinzuweisen, an welchen Stellen geschriebener Code bestimmte Richtlinien im Bereich Clean Code (siehe Kapitel 2.2.2) verletzt, sodass Lernende im Ansatz damit begonnen haben, ein Gefühl dafür zu entwickeln, wie sauberer Code geschrieben werden kann. Darüber hinaus konnten die automatischen Tests ein erstes Feedback liefern, dass von Lernenden genutzt werden konnte, um eine Aufgabe den Anforderungen entsprechend zu lösen. Infolgedessen konnten Probleme schnell identifiziert und nachher auch angesprochen werden. Dieser Zyklus lieferte somit formatives Feedback (siehe Kapitel 2.1.4), welches dazu genutzt werden konnte, Lernprozesse zu steuern.

Durch die Verteilung und Abgabe von Aufgaben über Gitlab wurden Lernende nicht nur an Technologien wie beispielsweise die Entwicklungsumgebung herangeführt, sondern konnten auch erste Erfahrungen mit Versionsverwaltung mittels Git und Gitlab machen. Schlussendlich stellt sich noch die Frage, ob mit diesem Konzept ein ähnlicher Lernerfolg auch mit weniger Betreuern beziehungsweise mehr Lernenden erzielt werden könnte. Dabei müsste beachtet werden, dass in einem solchen Fall viel weniger Betreuung und Feedback von Lehrenden möglich wäre.

7.2 Softwaretechnik 2

7.2.1 Beschreibung des Falls

Der Kurs *Softwaretechnik 2* an der *TH Köln Gummersbach* beschäftigt sich mit der Architektur von Software und Möglichkeiten, diese Architektur mithilfe von gängigen Werkzeugen in Programmcode umzusetzen (vgl. *Softwaretechnik 2 (ST2)* 2021). Durchschnittlich besuchen den Kurs etwa 120 Lernende. Im Sommersemester 2020 wurde erstmals Domain Driven Design (siehe Kapitel 2.2.1) verstärkt thematisiert und vermehrt Programmieraufgaben in das Praktikum integriert, welche in einem ersten Schritt automatisch getestet wurden. Viele Elemente wurden aus dem Kurs CE01 übernommen. So wurden Aufgaben wieder auf Gitlab hochgeladen, mithilfe von JUnit-Tests überprüft und die Ergebnisse auf einer Übersichtsseite dargestellt. Alle Tests auf der Übersichtsseite mussten bis zum Ende einer Deadline erfolgreich durchlaufen, damit das Praktikum als Bestanden galt.

Aufgrund der zeitgleichen *Corona-Pandemie* mussten allerdings weitere Anpassungen vorgenommen werden. Da Präsenztermine nicht mehr möglich waren, mussten Vorlesungen online über Zoom stattfinden. Da sämtliche Praktikumsaufgaben von Zuhause bearbeitet werden mussten, wurden diese mittels des in Kapitel 6 vorgestellten Divekits individualisiert und mit diesem automatisch auf Gitlab verteilt. Infolgedessen wurden Klassennamen, Attributnamen und auch Zustandsnamen variiert. Aufgaben waren also ähnlich genug, um sich mit anderen darüber auszutauschen, aber unterschiedlich genug, um ein einfaches Kopieren von Lösungen zu erschweren. Auch wenn die JUnit-Tests am Ende bestimmt haben, ob jemand bestanden hat, durften Betreuer über Discord (siehe Kapitel 4.2.2) nach Belieben angeschrieben werden, um Ratschläge oder Hilfestellungen zu erhalten. Wenn nötig, haben sich Lernende und Betreuer in einem Discord Channel eingefunden, um das Problem zu diskutieren oder gegebenenfalls auch eine Bildschirmübertragung zu starten, damit Betreuer einen tieferen Einblick in die Aufgabenbearbeitung hatten.

Nicht nur die Praktika, sondern auch die Klausur musste digital von Zuhause stattfinden. Folglich wurde auch die Klausur über Gitlab und Discord organisiert. Zusätzlich wurde die Klausur ebenfalls mithilfe des Divekits individualisiert und verteilt. Lernende hatten jeweils ein eigenes Projekt mit Klausuraufgaben. Für Fragen während der Klausur wurde ein extra Fragen-Channel in Discord eingerichtet, passende Antworten wurden in einen anderen Antworten-Channel eingestellt. Natürlich waren auch Fragen außerhalb von Discord über E-Mails oder Telefonate mit einem Betreuer erlaubt.

7.2.2 Feedback über Discord

Nach dem Praktikum und der Klausur wurde auf Discord Feedback diesbezüglich gesammelt. Im Folgenden wird auf das Feedback eingegangen, welches aus zwei Konversationen in Discord stammt (siehe Anhang 1.1):

Das Gesamtkonzept des Praktikums kommt bei den Lernenden gut an. Die Kommunikation über Discord wird positiv wahrgenommen. Es wird für gut befunden, dass Lernende nicht nur von Betreuern Hilfe erhalten haben, sondern dass sich Lernende auch gegenseitig über Discord helfen konnten. Auch die automatischen Tests werden geschätzt, weil durch diese schnelles Feedback möglich ist. Einige Lernende wünschen sich allerdings eine Aufgabenstellung, welche mehr Praxisbezug hat. Dies würde allerdings mit gewissen Schwierigkeiten einhergehen, wenn eine weitreichende Individualisierung weiterhin eine Rolle spielen soll. Von anderen würde *Kotlin* als Programmiersprache *Java* vorgezogen werden, da vorherige Kurse bereits auf Kotlin gesetzt haben. An dieser Stelle ist jedoch anzumerken, dass das Erlernen weiterer Programmiersprachen mit relativ wenig Arbeit verbunden ist, wenn schon gewisse andere Programmiersprachen bekannt sind. Zudem sollten Entwickler offen für weitere Sprachen sein und sich nicht auf eine Sprache beschränken. Schließlich zählt Java weiterhin zu den heutzutage relevanten Sprachen.

Das Feedback bezogen auf die Klausur umfasst viele Probleme, welche typisch für Klausuren sind. Auch hier wird das Konzept, welches viel Programmierung während der Klausur fordert, gut angenommen. Der Schwierigkeitsgrad und der Komplexitätsgrad der Klausuraufgaben wird als angemessen empfunden. Dennoch berichten einige Lernende von einem Zeitproblem. Dessen Hauptursache scheint darin zu liegen, dass einige Lernende die Aufgabenstellung an einer Stelle falsch verstanden haben, sodass diese bei der dementsprechenden Aufgabe mehr Programmcode implementiert haben als nötig. Um dies auszugleichen, wurde bei der Bewertung der Klausur darauf Rücksicht genommen, indem für richtig implementierte Zusatzlösungen Bonuspunkte verteilt wurden. Wie zu erwarten wird dieser Umstand positiv wahrgenommen. Für die Zukunft jedoch wünschen sich Lernende, dass Aufgabenstellungen klarer ausdrücken, was genau implementiert werden muss oder was beispielsweise nur spezifiziert werden soll. Als weiteren Grund für Zeitprobleme nennen Lernende den Umstand, dass mit mehreren Klausuraufgaben auch Kontextwechsel einhergehen. Diese scheinen ein schnelles Verstehen der Aufgaben zu behindern. Hauptgrund für diese Kontextwechsel sind die Individualisierungsverfahren, mit welchen Aufgabenvarianten erzeugt werden. Diese Varianten lassen sich einfacher definieren, wenn zumindest einzelne Aufgaben unabhängig voneinander betrachtet werden können.

7.2.3 Umfrage

Am Ende des Kurses wurde außerdem eine anonyme Online-Umfrage gestartet. Diese wurde 17-mal von Lernenden beantwortet, wobei einige Lernende bestimmte Fragen übersprungen haben. Die Ergebnisse der Umfrage sind in einer gekürzten Form dem Anhang zu entnehmen (siehe Anhang 1.2). Für diese Arbeit nicht relevante Fragen werden dabei nicht aufgeführt. Im Folgenden wird auf die Ergebnisse besonders interessanter Fragen eingegangen:

Das Prinzip der Online-Vorlesungen ist sehr gut angekommen, denn ein Großteil der Lernenden gibt an, dass sie mit dem Format der Online-Vorlesungen besser klargekommen sind, als mit der herkömmlichen Lehre mitsamt Präsenzterminen. Fast alle geben an, dass das Praktikum dabei geholfen hat, Vorlesungsinhalte nachzuvollziehen und zu verinnerlichen. Ein Grund dafür könnte sein, dass sich die Lernenden mit den Praktikumsaufgaben auseinandersetzen mussten, da aufgrund der Individualisierung ein einfaches Kopieren nicht möglich war. Zwei Drittel der Lernenden geben an, dass sie keine Teillösungen von anderen kopiert haben. Der Rest nennt als Gründe für das Kopieren von Teillösungen, dass sie zusammengearbeitet haben und Inhalte aufgeteilt haben. Ein Lernender nennt außerdem als Grund, dass Tests zu präzise waren und somit ein kleiner Fehler dazu führen konnte, dass ein Test nicht erfolgreich durchlief.

Alle Lernende bis auf einen empfinden die Wahl der Programmiersprache Java als gut bis sehr gut. Ähnlich verhält es sich auch zu der Kombination aus Git und Gitlab. Hier wird die Wahl von allen als gut bis sehr gut beurteilt. Etwas anders verhält es sich bei der Wahl des *Spring-Frameworks*. Diesbezüglich liegt ein breit gefächertes Meinungsbild vor. Ein möglicher Grund hierfür könnte die nötige Einarbeitungszeit in das Spring-Framework sein, welche sich ergibt, wenn bisher keinerlei Erfahrungen mit diesem gemacht wurden.

Ein Großteil der Lernenden gibt an, dass die Praktikumsabgabe über automatisierte Tests besser sei als herkömmliche mündliche Abgaben. Meistens waren Fehlermeldungen von Unit-Tests aussagekräftig und nachvollziehbar und erlaubten somit ein schnelles *Debugging*. Jenes wurde allerdings erheblich erschwert, wenn Tests versteckt waren, der Programmcode des Tests also nicht einsehbar war. Um ein frühes Kompilieren des Programmcodes zu gewährleisten, nutzten Tests *Java-Reflection*. Dies wirkte sich natürlich negativ auf die Lesbarkeit des Test-Programmcodes aus. Dennoch war dieser für einen Großteil der Lernenden verständlich. Lediglich zwei Lernende geben an, dass der Test-Programmcode eher weniger verständlich war.

Wenn Lernende bei der Bearbeitung der Aufgaben keinen Fortschritt machen konnten, weil das Feedback der automatischen Tests nicht ausreichte, wurde Feedback über Discord ermöglicht. Fast alle geben an, dass sie mit dem Kommunikationskanal Discord zur Betreuung viel zufriedener sind als mit herkömmlichen Kanälen wie beispielsweise Sprechstunden, E-Mails oder Foren. Dies könnte unter anderem durch das schnelle Feedback begründet sein, welches durch Discord ermöglicht wird. Zwei Drittel der Lernenden geben an, dass sie im Schnitt innerhalb von einer Stunde eine Hilfestellung erhalten haben.

7.2.4 Erkenntnisse

Aufgrund fehlender Präsenztermine und der relativ hohen Zahl an Lernenden war zwischen Lehrenden und Lernenden deutlich mehr Distanz zu spüren als es beispielsweise bei CE01 der Fall war. Durch das Nutzen von Discord als Kommunikationskanal außerhalb von Vorlesungen war es allerdings möglich eine Art virtuellen Klassenraum zu schaffen, wo Austausch in Bezug auf Lehrmaterial und Praktikumsaufgaben stattfinden konnte. Dies hat verhindert, dass die Distanz zwischen Lehrenden und Lernenden Ausmaße annimmt, welche sich negativ auf den Lernfortschritt auswirken könnten. Darüber hinaus war über Discord sehr schnelles Feedback möglich, weil die Plattform sowohl von Lehrenden als auch von Lernenden ausgiebig genutzt wurde. Auch während der Online-Klausur konnte innerhalb von Discord ein Ort geschaffen werden, an dem Fragen gestellt und Antworten von Lernenden bei Bedarf eingesehen werden können. Somit konnten außerdem Probleme schnell identifiziert und aus der Welt geschafft werden.

Durch Git-basierte Projekte als Träger der Aufgaben kamen Lernende außerdem vermehrt mit Git und Gitlab in Berührung, wobei vor allem Git ein extrem wichtiges Werkzeug zur Versionsverwaltung darstellt. Darüber hinaus wurden Lernende an das Konzept von automatischen Tests herangeführt und konnten somit lernen, Test-Programmcode zu lesen und gegen diesen zu entwickeln. Zugleich konnten durch die automatischen Tests betreuende Ressourcen entlastet werden, sodass Betreuer aufgrund zusätzlich verfügbarer Zeitressourcen qualitativ hochwertigere Hilfestellungen bieten konnten.

In Bezug auf Domain Driven Design konnte der Bereich des *Tactical Designs* gut in Praktikumsaufgaben thematisiert werden, auch wenn diese von der Komplexität nicht an zu lösende Probleme aus der Praxis heranreichen. Mithilfe der Technologien Java und Spring konnte aufgezeigt werden, inwiefern sich bestimmte Konzepte aus dem Domain Driven Design in Programmcode überführen lassen. Hierbei wurde in einem ersten Schritt ein *Rich Domain Model* bestehend aus Building Blocks nach dem Ansatz des Domain Driven Designs entworfen und mithilfe von *Spring Data JPA* implementiert. Anschließend wurde

die Domäne in Form von Aggregates strukturiert, welche nachher als *REST API* mit Hilfe von *Spring Web MVC* verfügbar gemacht wurden (vgl. *Softwaretechnik 2 (ST2)* 2021).

Wenn Lernende dazu gebracht werden sollen, sich mit dem Lehrmaterial zu beschäftigen, so hat die Individualisierung von Praktikumsaufgaben eine gute Alternative zu kontrollierten Umgebungen aufgezeigt, welche aufgrund fehlender Präsenztermine nicht möglich waren. Dies lässt sich ebenso auf die Online-Klausur anwenden, wo Individualisierung ein Kopieren von Lösungen erheblich erschwert, wenn beachtet wird, dass Lernende unter Zeitdruck stehen. Allerdings muss hier darauf geachtet werden, dass wechselnde Aufgabenkontexte leicht zu verstehen sind, indem Aufgaben knapp und präzise formuliert werden.

7.3 Softwaretechnik 1

7.3.1 Beschreibung des Falls

Der Kurs *Softwaretechnik 1* an der *TH Köln Gummersbach* beschäftigt sich zum einen mit dem Konstruieren eines Domänenmodells nach dem Ansatz Domain Driven Design auf Basis von gegebenen fachlichen Anforderungen. Zum anderen wird im Rahmen des Kurses das daraus resultierende Softwaresystem mithilfe von verschiedenen UML-Diagrammtypen spezifiziert und letztendlich in Form von Programmcode implementiert (vgl. *Softwaretechnik 1 (ST1)* 2021). Auch hier sind im Durchschnitt etwa 120 Lernende zu erwarten. Für den Kurs, welcher im Wintersemester 2020 stattgefunden hat, wurden einige Konzepte aus Softwaretechnik 2 übernommen. Praktikumsaufgaben wurden wieder mithilfe des Divekits individualisiert und anschließend auf Gitlab mitsamt automatischer Tests verteilt. Beigefügte Tests mussten dabei zum Bestehen des Praktikums erfolgreich durchlaufen.

Von Lernenden erstellte UML-Diagramme mussten allerdings von Betreuern auf ihre Richtigkeit überprüft werden. Wenn ein Lernender ein Diagramm zur Korrektur freigeben wollte, konnte dieser einfach eine kurze Nachricht in einen Discord-Channel schreiben. Bei Zeiten hat sich dann ein Betreuer die Lösung angeschaut und Feedback in einer dafür vorgesehenen Datei verfasst, sodass dieses Feedback mitsamt dem Feedback der automatischen Tests dargestellt werden konnte.

Die Kommunikation und Betreuung fand wieder über Zoom und Discord statt. Präsenztermine waren immer noch nicht möglich, das Format der Online-Vorlesungen wurde allerdings ein wenig überarbeitet. Wissen wurde nun nicht mehr allein über die Online-Vorlesungen vermittelt, sondern in Form von Videos vorab zur Verfügung gestellt. Anstelle

der Online-Vorlesungen haben Workshops über Zoom und Discord stattgefunden, welche nach dem Konzept Active Learning (siehe Kapitel 2.1.1) gestaltet waren. Die Aufgabenstellung und die Ergebnisse wurden dabei in Zoom diskutiert. Während der Bearbeitungszeit einer Aufgabe wurde jedoch zu Discord gewechselt. Dort konnten sich Lernende in Gruppen zusammenfinden und gemeinsam eine Lösung erarbeiten. Wenn sich Fragen oder Probleme ergeben haben, konnte in einem dafür vorgesehenen Discord Channel ähnlich wie bei Softwaretechnik 2 um Hilfe von Betreuern gebeten werden.

7.3.2 Erkenntnisse

Obwohl keine Präsenztermine möglich waren, hat das Workshop-Konzept gut funktioniert. Wie auch in Softwaretechnik 2 bildete sich innerhalb Discord eine Art virtueller Klassenraum. In den Übungsgruppen wurde diskutiert, gemeinsam an einer Lösung gearbeitet und sich gegenseitig Fragen beantwortet. Kam die Gruppe als solche nicht weiter, wurde um einen Betreuer gebeten, welcher nur einen Klick von der Gruppe entfernt war. Durch das Fördern von Gruppenarbeit während der Workshop-Aufgaben wurde die Gruppenarbeit häufig auch im Rahmen der verpflichtenden Praktikumsaufgaben fortgeführt. Dabei konnten sich einzelne Lernende jedoch nicht einfach an Lösungen bedienen, welche in der Gruppe entstanden sind, weil diese aufgrund der Individualisierung variierten. Die Lösung musste also zumindest in den eigenen Aufgabenkontext überführt werden, was ein gewisses Maß an Verständnis der Materie erforderte.

Das Kontrollieren der fertigen Diagramme durch Betreuer hat allerdings mehr Zeit in Anspruch genommen, als anfangs erwartet wurde. Dies lag vor allem daran, dass Lernende so oft abgeben konnten, wie sie wollten und daran, dass Lösungen nicht vorab gefiltert wurden. In den späteren Meilensteinen wurden deshalb strukturierte Markdown-Tabellentests vorgeschaltet, welche die Lernenden in die richtige Richtung lenken sollten. Beispielsweise wurde hier die richtige Benennung von Klassen überprüft oder geschaut, ob in Zusammenhang mit einem Zustandsdiagramm die richtigen Zustände identifiziert wurden. Die Tabellentests mussten dann erst erfolgreich durchlaufen, bevor ein Betreuer ein Diagramm korrigiert hat. Durch diese Maßnahme konnte bei der manuellen Korrektur einiges an Zeit eingespart werden.

8 Fazit

Im Rahmen dieser Arbeit wird die praxisorientierte Lehre von modernen Programmieransätzen aus verschiedenen Blickwinkeln betrachtet. Neben der Literatur werden außerdem die Erkenntnisse von durchgeführten Experteninterviews genutzt, um auf Basis dieser Empfehlungen zur Gestaltung und Organisation von modernen Programmierpraktika auszusprechen. In diesem Zusammenhang wird erst einmal erörtert, durch welche Merkmale sich moderne Programmierung definiert. Außerdem wird auf den Faktor Motivation der Lernenden eingegangen. Es wird betont, externe Regulierungen, welche der extrinsischen Motivation zuzuordnen sind, sparsam und nur da wo nötig einzusetzen. Ansonsten könnte die intrinsische Motivation der Lernenden negativ beeinflusst werden.

Darüber hinaus hat sich herausgestellt, dass die Programmieransätze Clean Code, Domain Driven Design und Test Driven Development in die Lehre integriert werden können, indem bestimmte Leitlinien beachtet und damit verbundene Anforderungen an die Rahmenbedingungen der Lehre angepasst werden. Bei dem Domain Driven Design sollte das Tactical Design im Fokus stehen, da das Strategic Design aufgrund mangelnder Zeitressourcen und einer oft nicht verfügbaren Fachseite in der Lehre schwer zu behandeln ist. Dahingegen können bei dem Tactical Design Glossare Abhilfe schaffen, um eine einheitliche Sprachbasis beziehungsweise Ubiquitous Language festzulegen. Zusätzlich hat sich im Verlauf der Fallstudie Softwaretechnik 2 gezeigt, dass eine Vielzahl von Building Blocks, welche dem Tactical Design zuzuordnen sind, im Rahmen der Lehre mithilfe des Spring Frameworks implementierbar und anhand diesem erklärbar sind. Infolgedessen kann der gelehrte theoretische Anteil des Tactical Designs auf konkrete Implementierungen erweitert werden. Wie das Domain Driven Design setzt auch das Test Driven Development fortgeschrittene Kenntnisse im Bereich der Programmierung voraus. Deshalb sollten Lernende früh gezielt mit Konzepten des Test Driven Developments konfrontiert werden, sodass der Ansatz im späteren Verlauf der Lehre besser aufgegriffen werden kann. Dazu gehört unter anderem das Entwickeln gegen Tests während der Bearbeitung von Praktikumsaufgaben, welche von Lehrenden bereitgestellt werden. Im Vergleich dazu lässt sich der Ansatz Clean Code sehr leicht in die Lehre integrieren. Wichtig diesbezüglich ist häufiges Feedback in Bezug auf Programmcode und das stetige Aufgreifen des Ansatzes über mehrere Kurse hinweg. Hierbei ist es allerdings wichtig, dass sich das Feedback nicht nur auf technische Feedbackmöglichkeiten beschränkt, sondern dass gegebenes Feedback auch von Lehrenden oder anderen Lernenden stammt.

Jedoch wird nicht nur in diesem Zusammenhang die Relevanz von häufigem schnellem Feedback für Lernende betont, sodass anschließend verschiedene Feedbackmöglichkeiten

diskutiert werden. Neben klassischem Feedback von Lehrenden über Face-to-face Kommunikation wird aufgezeigt, wie Lernenden zusätzliches Feedback über Instant-Messenger und automatische Tests gegeben werden kann. Diesbezüglich können Instant-Messenger wie beispielsweise Discord und ein auf Gitlab basierendes Testkonzept Abhilfe schaffen. Durch Instant-Messenger kann dabei ein virtueller Klassenraum geschaffen werden, welcher die Kommunikation zwischen Lehrenden und Lernenden und einzelnen Lernenden anregt, sodass ergänzende Feedbackkanäle geschaffen werden. Dahingegen generiert das Testkonzept Feedback auf einer technischen Basis und vermittelt dabei eine Wahrheit in Bezug auf eine Aufgabe, welche im Programmcode selbst liegt.

Letztendlich wird das Divekit als ein Werkzeug vorgestellt, welches das automatische Testkonzept unterstützt und außerdem Funktionalitäten zur Individualisierung von Aufgaben integriert. Hierbei können Aufgaben mithilfe der drei Individualisierungsarten Objekt-Individualisierung, Beziehungs-Individualisierung und Logik-Individualisierung variiert werden. Allerdings sollten je nach Aufgabenbereich jeweils andere Individualisierungsarten bevorzugt werden. Insbesondere die Beziehungs-Individualisierung eignet sich eher für Aufgaben, bei welchen ein ausführlicher fachlicher Aufgabentext nicht im Vordergrund steht, da die Formulierung eines solchen Textes sehr komplexe Ausmaße annimmt, wenn Beziehungen zwischen einzelnen Objekten variieren. Das Individualisierungskonzept richtet sich dabei nach den Anforderungen, welche sich aus der praxisorientierten Lehre von modernen Programmieransätzen ergeben, um das Kopieren von Lösungen anderer zu verhindern oder zumindest zu erschweren.

Mit Blick auf die Zukunft wäre eine natürliche Sprachunterstützung bei der Aufgabenerstellung mittels des Divekits von Vorteil, um ausführliche Aufgabentexte besser konzipieren und formulieren zu können. An dieser Stelle wären Weiterentwicklungen interessant, welche es erlauben, die Beziehungs-Individualisierung einfacher in den Prozess der Aufgabenerstellung zu integrieren. Auch wenn das Format zur Definition von Aufgabenindividualisierungen sehr vielseitig nutzbar ist, wären darüber liegende Schichten für dedizierte Anwendungsfälle hilfreich, um die Mechanismen zur Individualisierung in verschiedenen Aufgabenkontexten einfacher anzuwenden.

Im Rahmen der Fallstudien hat sich der Instant-Messenger Discord als Kommunikationsplattform bewährt. Allerdings, mussten die Veranstaltungen, auf welche dies zutrifft, aufgrund der Corona-Lage ausschließlich in einem digitalen Rahmen stattfinden. Interessant wäre es zukünftig, wenn Instant-Messenger wie Discord innerhalb eines hybriden Lehrkonzepts zusammen mit Präsenzterminen erprobt werden würden.

Literaturverzeichnis

Armstrong, Patricia (2016). *Bloom's taxonomy*. Vanderbilt University Center for Teaching.

Beck, Kent (2002). *Test-driven development: by example*. Addison Wesley.

Bradner, Scott (1997). *Key words for use in RFCs to Indicate Requirement Levels*. BCP 14. RFC Editor. URL: http://www.rfc-editor.org/rfc/rfc2119.txt.

Breitner, Joachim, Martin Hecker und Gregor Snelting (2017). "Der Grader Praktomat". In: *Automatisierte Bewertung in der Programmierausbildung*. Hrsg. von Oliver J. Bott u. a. Digitale Medien in der Hochschullehre 6. Waxmann Verlag GmbH, S. 159–172. URL: https://www.waxmann.com/automatisiertebewertung/.

Daft, Richard L. und Robert H. Lengel (1984). "Information richness: a new approach to managerial behavior and organization design". In: *Research in Organizational Behavior* 6, S. 191–233.

De Kok, Ties (2020). *Creating a collaborative virtual classroom: the case of Discord*. https://arc.eaa-online.org/blog/creating-collaborative-virtual-classroom-case-discord. Abgerufen: 17.02.2021.

Deci, Edward L, Richard Koestner und Richard M Ryan (1999). "A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation." In: *Psychological bulletin* 125.6, S. 627–668.

Dein Ort zum Reden (o.D.). https://discord.com/. Abgerufen: 17.02.2021.

Dokumentation für Autoren (o.D.). https://docu.ilias.de/ilias.php?ref_id=2221&cmd=layout&cmdClass=illmpresentationgui&cmdNode=iv&baseClass=ilLMPresentationGUI. Abgerufen: 04.02.2021.

Evans, Eric (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Felder, Richard M und Rebecca Brent (2009). "Active learning: An introduction". In: *ASQ higher education brief* 2.4, S. 1–5.

Forehand, Mary (2010). "Bloom's taxonomy". In: *Emerging perspectives on learning, teaching, and technology* 41.4, S. 47–56.

George, Boby und Laurie Williams (2004). "A structured experiment of test-driven development". In: *Information and software Technology* 46.5, S. 337–342.

GitLab Docs (o.D.). https://docs.gitlab.com/ee/. Abgerufen: 07.04.2021.

Goguadze, Giorgi (2010). *ActiveMath-generation and reuse of interactive exercises using domain reasoners and automated tutorial strategies*.

Hazzan, Orit, Noa Ragonis und Tami Lapidot (2020). *Guide to Teaching Computer Science: An Activity-Based Approach*. Springer Nature.

Helfrich-Schkarbanenko, Andreas u. a. (2018). *Mathematische Aufgaben und Lösungen Automatisch Generieren*. Springer.

Kaiser, Robert (2014). *Qualitative Experteninterviews: Konzeptionelle Grundlagen und praktische Durchführung*. Springer-Verlag.

Krinke, Jens, Maximilian Störzer und Andreas Zeller (2002). "Web-basierte Programmierpraktika mit Praktomat". In: *Softwaretechnik-Trends* 22.3.

Kurs Coding Essentials 1 (CS12) (o.D.). https://coco.study/kurse/110-coding-software-1/112-coding-essentials-1/. Abgerufen: 07.04.2021.

Li, Lan (2011). "How Do Students of Diverse Achievement Levels Benefit from Peer Assessment?" In: *International Journal for the Scholarship of Teaching and Learning* 5.2.

Mackinnon, Tim, Steve Freeman und Philip Craig (2000). "Endo-testing: unit testing with mock objects". In: *Extreme programming examined*, S. 287–301.

Martin, Robert C (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

Pfadenhauer, Michaela (2002). "Auf gleicher Augenhöhe reden". In: *Das Experteninterview*. Springer, S. 113–130.

Ras, Eric und Desirée Joosten-ten Brinke (2015). *Computer Assisted Assessment. Research into E-Assessment*. Springer.

Ridgway, Jim, Sean McCusker und Daniel Pead (2004). *Literature review of e-assessment*. Futurelab.

Ryan, Richard M und Edward L Deci (2000). "Intrinsic and extrinsic motivations: Classic definitions and new directions". In: *Contemporary educational psychology* 25.1, S. 54–67.

Skains, Lyle (2020). *Teaching on Discord*. https://medium.com/@lskains/teaching-on-discord-a96b510986b. Abgerufen: 17.02.2021.

Softwaretechnik 1 (ST1) (2021). https://www.archi-lab.io/display/public/ST1. Abgerufen: 07.04.2021.

Softwaretechnik 2 (ST2) (2021). https://www.archi-lab.io/display/public/ST2. Abgerufen: 07.04.2021.

SonarQube Documentation (o.D.). https://docs.sonarqube.org/latest/. Abgerufen: 30.01.2021.

Tavangarian, Djamshid u. a. (2004). "Is e-Learning the Solution for Individual Learning?." In: *Electronic Journal of E-learning* 2.2, S. 273–280.

Van Baarsen, Jeroen (2014). *GitLab Cookbook*. Packt Publishing Ltd.

Williams, Laurie, E Michael Maximilien und Mladen Vouk (2003). “Test-driven development as a defect-reduction practice”. In: *14th International Symposium on Software Reliability Engineering*, S. 34–45.

Anhang

1 Feedback zu Softwaretechnik 2

1.1 Feedback innerhalb Discord

1.1.1 Feedback zum Praktikum

[30.04.20 13:26] Betreuer 3
Hier dürft ihr gerne Feedback hinterlassen, wie ihr das neue Praktikumskonzept findet.

[30.04.20 13:26] Betreuer 3
Gibt auch keinen Punktabzug o.ä.

[30.04.20 13:31] Student 1
Das Konzept finde ich super. Ist schön zu sehen, dass hier auch von anderen Teilnehmern anstatt nur von Betreuern schnell geholfen wird. Meiner Meinung nach klappt die Art und Weise der Didaktik wunderbar und die automatischen Tests sind auch super, damit man schnell ein “Feedback“ bekommt. Was ich mir wünschen würde wäre eine etwas praxisnähere Aufgabenstellung (Mal Ehrlich: Wer modelliert einen Auspuff?).

[30.04.20 13:32] Student 2
Ich kann Student 1 in seinen ersten beiden Zeilen nur zustimmen.

[30.04.20 13:32] Betreuer 3
> Was ich mir wünschen würde wäre eine etwas praxisnähere Aufgabenstellung (Mal Ehrlich: Wer modelliert einen Auspuff?).

Kann ich verstehen. Ist ein bisschen der Tatsache geschuldet, dass das ganze randomisiert wird und dabei halt noch irgendwie Sinn ergeben muss.

[30.04.20 13:34] Student 3
Eventuell das Praktikum beim nächsten mal in Java und Kotlin anbieten, da die Leute, die das Fach in dem Semester machen, indem es vorgesehen ist und nicht in einem höheren Semester sind, seit AP2 im 2. Semester nur in Kotlin programmiert haben und Java nur im 1. Semester etwas in AP1 behandelt wurde, aber auch nicht sehr detailliert. Es ist auch nicht ein großer Umstieg von Kotlin zu Java aber dennoch ist es schade, wenn vorher Kotlin die ganze Zeit angepriesen wird, da es alles was Java macht nur in besser

macht und dann darauf leider nicht aufgebaut wird. Aber sonst finde ich die Umsetzung des Praktikums sehr gelungen.

[...]

[30.04.20 19:28] Betreuer 1
> Eventuell das Praktikum beim nächsten mal in Java und Kotlin anbieten, da die Leute, die das Fach in dem Semester machen, indem es vorgesehen ist und nicht in einem höheren Semester sind, seit AP2 im 2. Semester nur in Kotlin programmiert haben und Java nur im 1. Semester etwas in AP1 behandelt wurde, aber auch nicht sehr detailliert. Es ist auch nicht ein großer Umstieg von Kotlin zu Java aber dennoch ist es schade, wenn vorher Kotlin die ganze Zeit angepriesen wird, da es alles was Java macht nur in besser macht und dann darauf leider nicht aufgebaut wird. Aber sonst finde ich die Umsetzung des Praktikums sehr gelungen.

Java ist immer noch die Realität in der großen Mehrzahl der Projekte, in die Sie nach dem Studium gehen werden - der “Mainstream“ sozusagen. Das müssen Sie können. Darüber hinaus sollte man sich als Informatiker die Haltung zulegen “kann ich eine Sprache, dann lerne ich die nächste schnell“. Alle 2-3 Jahre kommt eine neue Sprache.

[30.04.20 19:46] Student 3
Ja das ist mir bewusst, dass man als Programmierer sich schnell anpassen können muss. Ich habe ja auch geschrieben, dass es kein großes Problem ist. Und ich bin mir auch im klaren, dass immer noch viel Java verwendet wird, auch wenn der Trend heutzutage weg von Java geht. Dachte nur es wäre eventuell eine Option Kotlin und Java anzubieten, wie in den meisten anderen Fächern. Vor allem, da man Java auch einfach in Kotlin umwandeln kann mit Intellij IDEA.

[30.04.20 19:50] Weiteres Teammitglied
Der Trend geht gefühlt seit 20 Jahren weg von Java. Bis jetzt hat es sich noch nicht bewahrheitet - man wird sehen.

[30.04.20 19:56] Student 3
das stimmt auch.

[01.05.20 12:39] Betreuer 1
> Dachte nur es wäre eventuell eine Option Kotlin und Java anzubieten, wie in den meis-

ten anderen Fächern. Vor allem, da man Java auch einfach in Kotlin umwandeln kann mit Intellij IDEA.

Dann wären aber vermutlich fast alle Studierenden bei Kotlin geblieben. Verstehen Sie mich nicht falsch, ich finde Kotlin super, bin auch ein großer Fan von anderen JVM-Sprachen mit weniger “Boilerplate“ als Java (z.B. Groovy). Es kann aber m.E. nicht angehen, dass Studierende hier ihren BA Informatik machen und kein Java programmiert haben. Daher Java als einzige Alternative. Daneben gibt es noch einen pragmatischen Grund: Bei einer Wahlfreiheit müssen wir das Tooling mit beiden Sprachen testen. Das kriegen wir zeitlich nicht hin.

[01.05.20 12:41] Student 3
Okay, das ergibt Sinn.

1.1.2 Feedback zur Klausur

[28.10.20 13:30] Student 1
Jetzt interessiert mich, wie’s bei euch lief.

[28.10.20 13:30] Student 2
Hab mich etwas erschlagen gefühlt.

[28.10.20 13:30] Student 3
Kam leider nicht ganz hin mit der zeit.

[28.10.20 13:31] Student 2
Also war ja alles in allem von den Aufgaben her vollkommen okay, aber ich hatte Probleme, mich da einzufinden in der Zeit.

[28.10.20 13:31] Student 2
Habe so oft den faden verloren von Aufgabe lesen, zu umsetzen.

[28.10.20 13:31] Student 2
Weils einfach so umfangreich war.

[28.10.20 13:31] Student 3
Ja same.

[28.10.20 13:31] Student 4
Ich kam auch nicht mit der Zeit aus.

[28.10.20 13:31] Student 5
Same, hatte leider kaum noch Zeit gehabt, um zu schauen, ob es funktioniert, was ich gecoded hab.

[28.10.20 13:32] Student 6
Selbes Problem...

[28.10.20 13:32] Student 7
Ohne 2 Bildschirme wäre das echt unübersichtlich geworden, war ziemlich viel hin-und-hergetabbe unter Zeitdruck, aber die Aufgaben an sich waren echt ok.

[28.10.20 13:33] Student 2
Für mich persönlich wars einfach sehr viel Kontext, den ich nicht schnell genug abgespeichert bekommen habe. Dementsprechend oft wieder nachlesen müssen und dann wieder aus dem Konzept gekommen.

[28.10.20 13:34] Student 8
Die Tabellenaufgabe war verwirrend, patch kam nie dran im Praktikum/ Vorlesung, das war verwirrend, und der ApplicationService sollte plötzlich Sachen machen, die im Praktikum ganz anders gelöst wurden (zumindest in meiner Lösung, aber da waren alle Tests grün).

[28.10.20 13:34] Student 8
Sonst war's halt viel, aber machbar von den aufgaben her.

[28.10.20 13:35] Student 1
> Ohne 2 Bildschirme wäre das echt unübersichtlich geworden, war ziemlich viel hin-und-hergetabbe unter Zeitdruck, aber die Aufgaben an sich waren echt ok.

Ich habe 3 Desktops verwendet (virtuelle).

[28.10.20 13:35] Student 9
Ich hatte extrem Probleme mit der Aufgabenstellung, ich habe aus versehen alle 7 Operation von der Rest Level 2 Aufgabe implementiert, weil ich "Unteraufgabe" als "extra"

wahrgenommen habe...

[28.10.20 13:35] Student 9
Well.

[28.10.20 13:35] Student 9
Shit.

[28.10.20 13:36] Student 8
Sollte man das nicht?

[28.10.20 13:36] Student 9
Idk.

[28.10.20 13:36] Student 1
Klar.

[28.10.20 13:36] Student 6
> Ich hatte extrem Probleme mit der Aufgabenstellung, ich habe aus versehen alle 7 Operation von der Rest Level 2 Aufgabe implementiert, weil ich “Unteraufgabe“ als “extra“ wahrgenommen habe...

Habe auch alle gemacht, da habe ich mich sehr lange dran aufgehalten.

[28.10.20 13:36] Student 10
Nur die mit (*) versehenen.

[28.10.20 13:36] Student 9
Im Nachhinein, wenn ich mir die Unteraufgaben durchlese, sieht es so aus, als sollte man die nur spezifizieren.

[28.10.20 13:37] Student 9
Aber ich habe die halt einfach mitgemacht.

[28.10.20 13:38] Student 1
Die Unteraufgaben waren Wegpunkte, wie kleine Meilensteine, falls man nicht weiß wo man anfangen soll, wenn ich das richtig verstanden habe.

[28.10.20 13:38] Student 6
Wow, danke, dann weiß ich, warum ich so lange gebraucht habe und mit der Zeit nicht klar kam...

[28.10.20 13:38] Student 2
Das mit den Unteraufgaben hat mich leider auch verwirrt.

[28.10.20 13:39] Student 11
Okay, mal gucken ob es für uns Bonuspunkte für die 4 extra Methoden geben wird.

[28.10.20 13:39] Student 8
Da waren Sternchen dran an den ersten drei (ich glaube weil man für die noch was in den Klassen hinzufügen musste), aber heißt das jetzt auch dass man nur die machen musste?

[28.10.20 13:39] Student 2
Von eher gutes Gefühl vor der Klausur zu Hoffen zu Bestehen.

[28.10.20 13:39] Student 9
> okay, mal gucken ob es für uns Bonuspunkte für die 4 extra Methoden geben wird.

Das dumme ist nur, dass ich dadurch die Controllerlogik am Ende gar nicht mehr machen konnte, wegen Zeit.

[28.10.20 13:39] Student 12
Alles in allem war es übersichtlich und gut machbar fand ich. Mir hat am Ende auch etwas Zeit gefehlt für Aufgabe 4.

[28.10.20 13:40] Student 1
Mit der Zeit, die wir jetzt hatten (2:30 Stunden) bin ich gerade so klar gekommen, musste aber ein paar Dinge im Ungewissen lassen. 2 Stunden wären für mich zu wenig gewesen, ich hatte so schon kaum Zeit, um die komplexeren Methoden und Logikbausteine zu testen. Ich verlass mich da zu 100 Prozent auf mein Gefühl. Zeit zum Unit-Tests schreiben hatte ich auch nicht, wäre mir wahrscheinlich auch zu viel geworden.

[28.10.20 13:40] Student 8
Würde gerade helfen wenn es mal eine aussage gäbe, ob man jetzt nur die 3 mit stern

oder alle implementieren musste.

[28.10.20 13:40] Betreuer 1
Nur die 3 mit stern.

[...]

[28.10.20 13:42] Betreuer 1
Text zu 2b: Im Package ... finden Sie “teilweise vollständige“ Klassen, **um die ersten drei Zugriffsoperationen zu implementieren (das sind die, die mit einem (*) gekennzeichnet sind)**.

[28.10.20 13:43] Student 11
Ja, an sich eindeutig formuliert, nur in der Eile überlesen.

[...]

[28.10.20 13:48] Betreuer 1
Ich bin für Vorschläge dankbar, wie man so etwas zukünftig noch deutlicher kennzeichnen kann. Ist immer ein Trade-Off zwischen “zu viel Text / unübersichtliche Aufgabenstellung“ und “klar und unmissverständlich ausgedrückt“.

[28.10.20 13:48] Student 1
Sie sagten mir ja auch, dass die Lösung die ich versehentlich gepostet hatte, die Richtige ist (indirekt). Also musste man das ja scheinbar implementieren. Allerdings war die 7 ja nicht Teil der ersten drei. Dachten Sie, dass die 7 mit einem Stern markiert war?

[28.10.20 13:49] Student 10
Was mMn. ungemein helfen würde wäre über die 4 Aufgaben hinweg einen gemeinsamen Kontext zu schaffen, sodass man sich nur einmal in die Aufgabenstellung herein denken muss und nicht bei allen vier Aufgaben nochmal neu. Würde auch dabei helfen, wenn man nochmal über die Aufgaben drüber geht, da würde vielleicht so etwas auffallen.

[28.10.20 13:49] Student 8
> Das API muss die folgenden Operationen unterstützen:

Der Satz klang leider sehr eindeutig im Eifer des Gefechts.

[28.10.20 13:49] Betreuer 1
> Sie sagten mir ja auch, dass die Lösung die ich versehentlich gepostet hatte, die Richtige ist (indirekt). Also musste man das ja scheinbar implementieren. Allerdings war die 7 ja nicht Teil der ersten drei. Dachten Sie, dass die 7 mit einem Stern markiert war?

Sie hatten keine Aufgabennummer dazu geschrieben. Hätte ich gewusst, dass Sie das implementieren, hätte ich was gesagt.

[28.10.20 13:50] Student 1
Okay, ich hatte vermutet Sie wüssten, welche Aufgabe gemeint ist.

[28.10.20 13:50] Betreuer 1
> Was mMn. ungemein helfen würde wäre über die 4 Aufgaben hinweg einen gemeinsamen Kontext zu schaffen, sodass man sich nur einmal in die Aufgabenstellung herein denken muss und nicht bei allen vier Aufgaben nochmal neu. Würde auch dabei helfen, wenn man nochmal über die Aufgaben drüber geht, da würde vielleicht so etwas auffallen.

Vermutlich - aber das ist ein echt unlösbares Problem, wenn man die Aufgaben individualisieren muss.

[28.10.20 13:51] Betreuer 1
> Okay, ich hatte vermutet Sie wüssten welche Aufgabe gemeint ist.

JPA ergänzen musste man bei 1, 2 und 4.

[28.10.20 13:51] Student 9
Also was mir allein schon geholfen hätte wäre eine klare Unterteilung in “Beschreibung des Szenarios“ und “Das müssen sie machen“. Ich hab den Part über “Unteraufgaben“ leider als Aufgabe interpretiert und dementsprechend versucht, alles umzusetzen bevor es an die Unteraufgaben geht, was in den ersten 3 auch noch funktioniert hat. Das ist natürlich ein Fehler meinerseits, aber für nächstes mal.

[28.10.20 13:51] Student 6
> Ich bin für Vorschläge dankbar, wie man so etwas zukünftig noch deutlicher kennzeichnen kann. Ist immer ein Trade-Off zwischen “zu viel Text / unübersichtliche Aufgabenstellung“ und “klar und unmissverständlich ausgedrückt“.

Gute Frage, vielleicht wie sie es gerade auch im Discord gemacht haben, in dick markieren. Oder die zu machenden Zugriffsoperationen einzeln auflisten.

[28.10.20 13:51] Student 13
War alles machbar, nur unter Zeitdruck nochmal was anderes.

[28.10.20 13:52] Student 6
Das stimmt, da so viel zeit bei Aufgabe 2 weggefallen ist, hatte ich echt Stress am ende.

[28.10.20 13:53] Student 2
Ja, zu schwierig war es nicht, war vollkommen angemessen. Aber ein paar Dinge haben verwirrt, dazu der Zeitdruck und viel/wechselnder Kontext hat es für mich etwas wild gemacht.

[28.10.20 13:53] Student 5
Stimme zu.

[28.10.20 13:53] Betreuer 1
Viele wirklich sinnvolle Ideen, danke.

[28.10.20 13:54] Betreuer 1
Die wechselnden Kontexte kriegt man leider nicht weg, die sind Tooling-immanent - sonst kann man die Aufgaben nicht individualisieren.

[28.10.20 13:54] Betreuer 1
Scheint aber ja insgesamt ein Zeit/Mengen-Problem zu geben.

[28.10.20 13:54] Student 11
Und nicht auf die eine Aufgabe fokussiert noch der Vorschlag Negierungen in den Aufgabenstellungen zu vermeiden um das Verständnis zu erleichtern, da erinnere ich mich gerade vor allem an die Aufgabe mit den Aggregat-Grenzen, wo man ziemlich einfach das “nicht“ hätte weglassen können.

[28.10.20 13:55] Student 9
> Scheint aber ja insgesamt ein Zeit/Mengen-Problem zu geben.

Ich denke, es wäre zwar etwas knapp geworden, wenn ich es nicht missverstanden hätte aber machbar.

[28.10.20 13:56] Student 8
Aber das Zeit/Mengen Problem ist ja keins, wenn sich bei der Korrektur herausstellt, dass die meisten nicht fertig geworden sind, werden doch die Punkte gesenkt, oder?

[28.10.20 13:58] Betreuer 1
Wir schauen uns das in der Summe an und werden dann angemessen handeln, wenn nötig. Man sieht das ganz gut in der Punkteverteilung. Ist bei mehr oder weniger jeder Klausur so.

[...]

[03.02.21 09:47] Betreuer 1
Hier ist nochmal ein Update zu ST2 - das Prüfungsamt war so freundlich, die Anmeldefrist für die Klausur zu verlängern, bis 19.2. Ich bin sehr zuversichtlich, dass wir die Ergebnisse bis Mitte nächster Woche haben, aber dann haben Sie schon mal die Sicherheit, dass Sie sich in jedem Fall neu anmelden können, wenn nötig.

[10.02.21 18:07] Student 8
Danke für die Transparenz, top gemacht. Auch das berücksichtigen der Nr. 2 bei denen die die falsch verstanden haben. Klasse.

1.2 Umfrage

Frage: Wie gut kamen Sie mit dem Format der Onlinevorlesungen (im Vergleich zur Präsenzlehre “vor Corona“) zurecht?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
deutlich schlechter	0	0%
etwas schlechter	1	7,14%
genauso gut	4	28,57%
etwas besser	3	21,43%
deutlich besser	6	42,86%

Frage: Hat das Praktikum dabei geholfen, Vorlesungsinhalte nachzuvollziehen, zu verstehen und zu verinnerlichen?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
leider gar nicht	0	0%
eher wenig	1	7,69%
war ok	0	0%
gut	4	30,77%
sehr gut	8	61,54%

Frage: Haben Sie Teillösungen oder Lösungen von anderen kopiert oder abgewandelt, um diese nicht selbst lösen zu müssen?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
nein	8	66,67%
ja	4	33,33%
dazu möchte ich nichts sagen	0	0%

Frage: Waren Sie mit der Wahl, die Programmiersprache Java für das Praktikum zu nehmen, zufrieden?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
leider gar nicht	1	8,33%
eher wenig	0	0%
war ok	0	0%
gut	6	50%
sehr gut	5	41,67%

Frage: Wie gut kamen Sie mit dem Spring Framework zurecht?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
leider gar nicht	0	0%
eher wenig	3	25%
war ok	3	25%
gut	2	16,67%
sehr gut	4	33,33%

Frage: Wie gut kamen Sie mit der Kombination aus Gitlab und Git zur Meilensteinabgabe zurecht?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
leider gar nicht	0	0%
eher wenig	0	0%
war ok	0	0%
gut	4	33,33%
sehr gut	8	66,67%

Frage: Praktikumsmeilensteine wurden automatiert in Form von Unit Tests abgenommen. Wie gut fanden Sie dies im Vergleich zu herkömmlichen mündlichen Praktikumsabnahmen vor Ort?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
leider gar nicht	0	0%
eher wenig	2	16,67%
war ok	1	8,33%
gut	5	41,67%
sehr gut	4	33,33%

Frage: Wie aussagekräftig und nachvollziehbar waren die Fehlermeldungen der Unit Tests?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
leider gar nicht	0	0%
eher wenig	2	16,67%
war ok	1	8,33%
gut	6	50%
sehr gut	3	25%

Frage: Bei manchen Unit Tests waren nur Error-Meldungen, allerdings nicht der Code des Unit Tests selbst einsehbar. Wie gut war Debugging unter diesem Umstand möglich?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
leider gar nicht	1	8,33%
eher wenig	3	25%
war ok	6	50%
gut	2	16,67%
sehr gut	0	0%

Frage: Die Unit Tests aus dem Praktikum nutzten oft Java - Reflection. Wie lesbar und verständlich war der Code der einsehbaren Unit Tests?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
leider gar nicht	0	0%
eher wenig	2	16,67%
war ok	5	41,67%
gut	4	33,33%
sehr gut	1	8,33%

Frage: Wie zufrieden waren Sie mit dem Kommunikationskanal Discord für die Betreuung des Praktikums (im Vergleich zu anderen Formen wie z.B. festen Abnahmeterminen in Praktikumsgruppen, Sprechstunden, Austausch per Email, ILIAS-Forum etc.)?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
viel schlechter	0	0%
etwas schlechter	1	8,33%
etwa ähnlich	0	0%
etwas besser	1	8,33%
viel besser	10	83,33%

Frage: Wie zeitnah konnte Ihnen bei Problemen geholfen werden?

Auswahloption	**Anzahl der Auswahlen**	**Prozentualer Anteil**
gar keine (brauchbare) Hilfe	1	8,33%
$>$ 3 Tage bis zu einer Hilfestellung	0	0%
1 - 3 Tage bis zu einer Hilfestellung	0	0%
$<$ 1 Tag bis zu einer Hilfestellung	3	25%
(fast) sofort, $<$ 1h	8	66,67%

2 Experteninterviews

2.1 Interviewleitfaden

2.1.1 Analysedimensionen ermitteln

Forschungsfragen	**Analysedimensionen**
Was macht *modernes Coding* aus, und wie kann man die Aspekte Basiswissen Coding, moderne Sprachen, DDD, Clean Code und TDD am besten in digitalisierbare Praxisübungen umsetzen? Was findet man in Literatur und Praxisangeboten? Was sind typische Lernziele?	• Organisation von digitalen Praxisübungen • Modernes Coding und damit verbundene typische Lernziele
Wie können solche praktischen Aufgaben für Lernende am besten in einem Maß individualisiert werden, dass ein einfaches Kopieren von Lösungen nicht möglich ist? Wie müssen dazu passende Werkzeuge aussehen, welche die Anforderungen und Erwartungen von Lehrenden und Lernenden erfüllen?	• Individualisierung • Werkzeuge
Mit welchen Prinzipien kann der Zielkonflikt aus automatisierter Überprüfbarkeit der Lösungen vs. Freiheit der Studierenden bei der Aufgabenbearbeitung weitestgehend neutralisiert werden?	• Automatisierte Aufgabenüberprüfung

2.1.2 Fragenkomplexe ermitteln

Analysedimensionen	**Fragenkomplexe**
Modernes Coding und damit verbundene typische Lernziele	• Umgang mit Werkzeugen zur Entwicklung von Code • Charakteristik von modernem Coding • Moderne Programmiersprachen • Moderne Konzepte und Herangehensweisen
Organisation von digitalen Praxisübungen	• Kooperation bei der Aufgabenbearbeitung (Einzelarbeit, Gruppenarbeit, Pair-Programming) • Motivation der Lernenden
Individualisierung	• Individualisierungsmethoden • Charakteristik von individualisierten Aufgaben (Schwierigkeit, Abweichungen) • Mehraufwände bei der Aufgabenerstellung

Automatisierte Aufgabenüberprüfung	• Kriterien für automatisierte Überprüfbarkeit • Prinzipien zur Gestaltung von automatisch überprüfbaren Aufgaben
Werkzeuge	• Automatische Aufgabenindividualisierung • Automatische Überprüfung von Aufgaben • Automatische Verteilung und Abgabe von Aufgaben • Feedback und Beratung

2.1.3 Interviewfragen ermitteln

Fragenkomplexe	Interviewfragen
Umgang mit Werkzeugen zur Entwicklung von Code	• Was sind Ihrer Meinung nach die wichtigsten Werkzeuge, welche im Rahmen von modernem Coding benötigt werden und deshalb gelehrt werden sollten? • Finden Sie es sinnvoller, wenn Lernende Werkzeuge zur Versionsverwaltung direkt über eingebaute Funktionen der Entwicklungsumgebung aufrufen, oder sollten Lernende die Versionsverwaltung über die Konsole ansteuern, um die genaue Funktionsweise dieses Werkzeugs besser durchdringen zu können?
Charakteristik von modernem Coding	• Was macht aus Ihrer Sicht modernes Coding aus?

Moderne Programmiersprachen	• Welche Programmiersprachen vereinen viele Paradigmen, welche für modernes Coding typisch sind? • Was würden Sie in der Lehre höher priorisieren: Ein Fokus auf weit verbreitete Programmiersprachen oder ein Fokus auf Programmiersprachen, welche auf modernen Paradigmen aufbauen und das Potential aufweisen, auch einmal eine breite Anwendung zu finden?

Moderne Konzepte und Herangehensweisen	• Wie kann man Lernende dazu bringen, sich an bestimmte Richtlinien zu halten, welche zu sauberen Code, also Clean Code führen? • Können Sie praktikable Wege nennen, mit dessen Hilfe diese Richtlinien in Übungsaufgaben erzwungen werden können, ohne dass dies über eine manuelle Korrektur erfolgen muss? • Welche konkreten Richtlinien wären Ihrer Meinung nach besonders wichtig zu überprüfen? • Domain Driven Design stellt eine Herangehensweise zur Modellierung von komplexer Software dar, wobei sich dieser Ansatz stark an der zu modellierenden Fachlichkeit orientiert. Was muss man bei der Erstellung von Übungsaufgaben beachten, welche diesen Ansatz lehren sollen, obwohl diese meist ein geringes Maß an Komplexität aufweisen?

Kooperation bei der Aufgabenbearbeitung (Einzelarbeit, Gruppenarbeit, Pair-Programming)	• Würden Sie eine Bearbeitung von Programmieraufgaben eher in Gruppen oder in Einzelarbeit bevorzugen, wenn diese dazu gedacht sind, bewertet zu werden? • Die Arbeitstechnik Pair-Programming beschreibt ein Verfahren, bei dem zwei Personen abwechselnd jeweils Programmieren und Feedback geben. Welche besonderen Vorteile Sehen Sie hier in Abgrenzung zur Gruppenarbeit?
Motivation der Lernenden	• Um Lernenden den nötigen Druck bei der Aufgabenbearbeitung zu machen, gibt es verschiedene Ansätze. Zum einen kann Druck durch Noten und Aufgabenindividualisierung geschaffen werden und zum anderen kann ein motivierendes Lernklima schon ausreichen. Unter welchen Umständen sollte Ihrer Meinung nach welcher Ansatz bevorzugt werden?
Individualisierungsmethoden	• Welche Individualisierungsmethoden sind Ihnen bekannt, welche Aufgaben in einer Art individualisieren, dass ein Kopieren von Lösungen erschwert wird?

Charakteristik von individualisierten Aufgaben (Schwierigkeit, Abweichungen)	• Was ist aus Ihrer Sicht wichtiger, ein hoher Grad an Individualisierung oder eine konsistente Geschichte, welche den Kontext der Übungsaufgabe beschreibt? • Wie ist Ihre Meinung zu abweichenden Schwierigkeitsgraden einzelner individualisierter Aufgaben aufgrund der Individualisierung?
Mehraufwände bei der Aufgabenerstellung	• Wie viel Mehraufwand wären Sie bereit zu investieren, für die Erstellung von Aufgaben, welche individualisiert werden können?
Kriterien für automatisierte Überprüfbarkeit	• Was muss eine Übungsaufgabe für Eigenschaften aufweisen, damit diese automatisch überprüfbar ist?
Prinzipien zur Gestaltung von automatisch überprüfbaren Aufgaben	• Haben Sie Ideen, wie Übungsaufgaben sowohl automatisch überprüfbar wären, als auch mit einer gewissen Freiheit in der Aufgabenbearbeitung zu lösen wären?

Automatische Aufgabenindividualisierung	• Haben Sie Werkzeuge zur Aufgabenindividualisierung schon einmal in der praxisorientierten Lehre angewendet und welche Erfahrungen konnten Sie daraus mitnehmen? • Welche Werkzeuge sind Ihnen bekannt, mit deren Hilfe sich Aufgaben in großem Stil individualisieren lassen?
Automatische Überprüfung von Aufgaben	• Inwieweit denken Sie, kann Feedback von automatischen Aufgabenkorrekturen das Feedback von manuell korrigierenden Lehrenden ersetzen?
Automatische Verteilung und Abgabe von Aufgaben	• Mit welchen Plattformen haben Sie bisher Erfahrungen gemacht, über die sich das Verteilen von Übungsaufgaben und das Abgeben von Lösungen realisieren lässt?
Feedback und Beratung	• Was halten Sie von modernen Messengern als Kommunikationsplattformen zum Austausch zwischen Lernenden und Lehrenden über Übungsaufgaben?

2.2 Interviewtranskripte

2.2.1 Experteninterview 1

Interviewpartner: Interviewter 1
Tätigkeitsfeld: Wissenschaftlicher Mitarbeiter
Datum: 17.12.2020
Ort: Zoom-Meeting

00:00–40:34

I: [00:01] Was sind deiner Meinung nach die wichtigsten Werkzeuge, welche man im Rahmen von modernem Coding benötigt und deshalb auch lehren sollte?

B: [00:13] Also es kommt immer darauf an, was man darunter versteht.

I: [00:23] Ich meine die essenziellen Werkzeuge, die man zum Coden braucht.

B: [00:28] Ja, die Entwicklungsumgebung, klar. Wobei da würde ich eher sagen, mir fällt da Maven ein. Das ist komplett unabhängig von der Entwicklungsumgebung. Es wäre sinnvoll, wenn man den Lernenden da die Wahl lassen würde. Im Rahmen des Hochschulstudiums wäre es natürlich sinnvoll, wenn man sich auf eine Entwicklungsumgebung einigen würde. Weil dann ist die Betreuung viel einfacher. Wenn man z.B. sagen würde, man nimmt Eclipse oder was auch immer, dann kann man auch besser Leitfaden schreiben und auch die Lernenden besser unterstützen. Aber im Prinzip wäre ein offenes Framework eine gute Wahl, welches unabhängig von der IDE ist. Im Java Umfeld ist das klassisch Maven. Also du brauchst jetzt nicht unbedingt eine IDE. Es reicht auch eine Konsole zum entwickeln. Wichtig meiner Meinung nach ist auch Continuous Integration, sei es jetzt Bamboo oder Jenkins, dann natürlich auch Versionsverwaltung z.B. Git.

I: [01:42] Das heißt, du würdest so etwas wie Build Umgebungen auch sehr früh lehren?

B: [01:48] Ja auf jeden Fall. Also das ist das, was mir permanent auffällt, auch hier im Studium. Die Studierenden lernen nicht, am Ende auch ein fertiges Produkt auszuliefern. Also es werden oft Schnipsel gecoded, vielleicht paar Tests dazu, aber so wirklich von Null auf zu sagen, dass man mal ein Produkt aufsetzt und dieses ausliefert, gehört einfach dazu. Und auch so etwas wie einen Jenkins aufzusetzen. Also über Infrastructure lässt sich drüber streiten? Muss man das jetzt machen in der heutigen Zeit oder nutzt man

irgendwelche Cloud Lösungen? Aber ich finde das gar nicht verkehrt, wenn man das auch selber aufsetzen könnte.

I: [02:41] Dann würde ich mal zur nächsten Frage übergehen. Findest du es sinnvoller, wenn man Lernende Werkzeuge zur Versions Verwaltung direkt über Funktionen der IDE bedienen lässt? Oder sollten die Lernenden die Konsole benutzen?

B: [02:58] Beides würde ich sagen. Ich würde erstmal weg von der IDE gehen und sagen, lernt erstmals tatsächlich mithilfe der Konsole Versionsverwaltung zu nutzen. Dann versteht man auch, was da passiert. Die IDE macht ja nichts anderes, nur dass man die Befehle nicht kennt. Ich meine, ich bin ja jetzt ein bisschen älter und ich habe damals auch C++ selber kompiliert und Make-Files geschrieben und klar, die IDE übernimmt viele Tätigkeiten für dich. Und das funktioniert auch ganz gut. Also auch wenn man im Java Umfeld programmiert und ein Maven Projekt importiert in die IDE, dann ist alles schon vorkonfiguriert. Der Class Path ist gesetzt und so weiter. Alle Libraries sind schon irgendwie angebunden und man bekommt gar nicht mit, was da passiert im Hintergrund. Also im Prinzip wie eine Black-Box. Schön und gut, aber wenn man das dann irgendwann mal selber machen muss und nachvollziehen muss, was wie zusammenhängt, dann bekommt man das nicht mit. Und daher würde ich sagen am Anfang auf jeden Fall die Basics selber kennenlernen. Und später, wenn man das z.B. im Berufsleben einsetzt und das schneller geht mit einer IDE, dann würde ich sagen, dann macht es schon Sinn, auch die Features, welche die IDE bietet, auch einzusetzen.

I: [04:20] Mal eine ganz generelle Frage. Was macht für dich modernes Coding aus?

B: [04:29] Für meine Begriffe macht modernes Coding aus, dass man sich nicht irgendwie auf eine Programmiersprache oder auf bestimmte Tools konzentriert, sondern auch mal links und rechts guckt und schaut, was gibt es aktuell auf dem Markt? Wie kann man etwas beschleunigen? Wo kann man irgendwelche Frameworks einsetzen, welche dir zum Teil die Arbeit abnehmen? Was sind die aktuellen Standards? Das eine ist die Frage, wie komme ich schnell zu einer Lösung? Da kann ich dann bestimmte Libraries einsetzen, um Code generieren zu lassen, der aber auch qualitativ hochwertig ist. Time to Market ist jetzt ein Thema. Sei es auch Continuous Integration, was bewirkt, dass ich nicht immer permanent alles lokal kompilieren und testen muss. Damals hatten wir diese Tools nicht gehabt. Man musste permanent alles bauen und lokal testen und wenn da mal ein Bug war, musste man den erstmal fixen. Heute hat man einen Developer Branch, auf den man committen kann und im Hintergrund wird dann gebaut und getestet, sodass man einfach

weitercoden kann. Aber im Wesentlichen bedeutet modern, dass man aktuelle Technologien verwendet und auch links und rechts schaut. Wenn man aber schnell produktiv gehen möchte, dann würde man eher auf erprobte und bewährte Technologien setzen, wo auch die Unterstützung da ist. Aber wie gesagt, man sollte auch links und rechts schauen und im Einzelfall entscheiden, was die richtige Technik oder Methodik ist.

I: [06:40] Wenn man sich jetzt mal eine Ebene darunter anschaut. Was für Programmiersprachen fallen dir ein, die auch so moderne Paradigmen vereinen?

B: [06:50] Darunter würde Go fallen. Welches aber im Vergleich zu Java viel weniger Unterstützung bietet. Also es gibt schon viele Open Source Projekte dazu und Frameworks. Diese werden aber oft halbherzig betreut. Also klar, Java ist da unschlagbar. Eine andere Programmiersprache wäre da noch Kotlin. Wobei das sehe ich so ähnlich wie bei der Frage mit der Versionsverwaltung. Da würde ich auch sagen, wir haben erstmal Java als Programmiersprache und dann lernt man später auch die Vorteile von Kotlin. Weil da ist auch vieles Magie, was man auf den ersten Blick nicht sieht.

I: [07:42] Nochmal eine andere Frage zu Programmiersprachen. Was würdest du in der Lehre höher priorisieren. Ein Fokus auf weit verbreitete Programmiersprachen oder ein Fokus auf Programmiersprachen, welche auf modernen Paradigmen aufbauen und das Potential aufweisen, auch einmal eine breite Anwendung zu finden?

B: [07:59] Beides. Also hauptsächlich vor allem hier an der Technischen Hochschule bilden wir ja schon Handwerker aus, welche nach einem abgeschlossenen Studium auch meistens coden können. Und da gehört dazu, dass man aktuell gängige Programmiersprachen kennenlernt. Aber man sollte natürlich auch Fokus auf Neuerungen haben. Es gibt bestimmte Programmiersprachen, Konzepte oder Frameworks, die für bestimmte Anwendungen eher geeignet sind als andere. Die sollte man auch kennenlernen.

I: [08:33] Das heißt, du würdest den Fokus sehr verteilt setzen und nicht explizit auf eines der beiden?

[08:41] Auf jeden Fall verteilt und wichtig ist, dass man tatsächlich ein Handwerk beherrscht. Dass man mit dem Studium fertig ist und sagen kann, dass man jetzt auch mit einer bestimmten Programmiersprache oder bestimmten Methodik und Konzepten wirklich ein Produkt am Ende umsetzen kann. Dieses auch zu deployen und nutzbar zu machen. Das Wesentliche besteht ja darin, sich weiterzubilden. Also wenn ich zurückdenke

an meine Zeiten. Als ich damals angefangen hatte mit Versionsverwaltung, war das damals noch CVS. Dann kam Subversion, dann kam Mercure. Dann kam Git und so weiter. Und diese Lernkurve wird sehr flach, wenn man davor schon die Konzepte kennenlernt. Also so stark unterscheiden die sich nicht. Also der wesentliche Vorteil von Versionsverwaltung ist dann klar. Da kommen ein paar Features hinzu, der Code liegt mittlerweile eben verteilt und nicht mehr zentral vor. Und das ist meiner Meinung nach das Wichtigste, dass man auch lernt, sich weiterzubilden und die Grundkonzepte kennenlernt.

I: [09:59] Gehen wir mal zu einem etwas tieferen Bereich über, und zwar: Viele Lernende können ja noch nicht gut coden und da entsteht manchmal Code, den man vielleicht nicht als sehr lesbar bezeichnen würde. Und jetzt die Frage, wie kann man denn die Lernenden dazu bringen, sich an bestimmte Richtlinien zu halten, die zu sauberen Code oder auch Clean Code führen?

B: [10:27] Ja, es gibt Style-Guides, die auch teilweise de facto Standard sind, welche man sich anschauen kann. Ich würde dann viele Beispiele liefern, diese erklären und auch Gegenbeispiele zeigen. Ich beobachte, dass viele auch nicht mit fremdem Code klarkommen. Also das muss man auch beibringen, auch mal Code zu lesen und zu überarbeiten und nicht nur eigene Lösungen zu entwickeln. Ich denke mal, das würde auf jeden Fall helfen. Und wenn man dann ein schlechtes Beispiel vorstellt und sagt, da existiert ein Bug und der soll gefixt werden, dann kann man daraus auch Lehren ziehen. Wenn der Code eben schlecht umgesetzt wurde und nicht lesbar ist, dann werden die Studenten schon selber draufkommen, wo das Problem liegt.

I: [11:26] Welche praktikablen Wege würden dir einfallen, wenn man die Studenten dazu zwingen will, sich an diese Richtlinien zu halten? Also das nicht nur einfach gesagt werden soll, dass man sich daranhalten soll, sondern das irgendwie auch erzwingen will.

B: [11:39] Da gibt es ja zahlreiche technische Lösungen, die das auch schon unterstützen z.B. für Jenkins gibt es Plugins, mit denen man Code Check-Styles durchführen kann. Also ich hatte ein Open-Source-Projekt im Bereich Smart Room betreut, da haben wir auch ein Binding entwickelt für Open Hub und wir hatten auch automatisierte Code Überprüfung, sei es Länge der Zeilen usw. Das gab es als Vorgabe und das konnte man in der IDE einbinden und automatisch durchchecken lassen. Das ist so ähnlich wie Tests, welche erstmal durchlaufen werden müssen und dann wird Code eventuell nicht akzeptiert. Das wäre eine Möglichkeit, die relativ einfach umzusetzen ist.

I: [12:39] Was wären denn konkrete Richtlinien, die du besonders wichtig finden würdest?

B: [12:52] Ich würde da jetzt wirklich auf die gängigen de facto Standards beim Code Style verweisen. Wichtig ist vielleicht einheitliche Sprache zu nutzen, dass da jetzt nicht z.B. Englisch und Deutsch vermischt wird. Dann generell auch, dass Code auch dokumentiert ist, wo es nötig ist. Wichtig ist auch, wie die Methodennamen gewählt sind, wie generell die Struktur von dem Projekt aussieht und so weiter.

I: [13:30] Ich habe noch eine Frage zu Domain Driven Design, und zwar ist das ja eine Herangehensweise zur Modellierung von meist sehr komplexer Software. Und wir lehren das ja auch bei uns. Die Frage ist, was muss man da beachten, wenn man das in Übungsaufgaben verpackt, die ja oft von der Komplexität eher geringer ausfallen?

B: [13:52] Also, wenn man zu große Aufgaben wählt, die zu komplex sind, dann läuft man die Gefahr, dass die Studenten damit nicht klarkommen oder die Zeit einfach nicht ausreicht. Wenn man die Aufgaben zu einfach gestaltet, dann kommt das eigentliche Lernziel nicht zum Tragen. Da muss man eine Lösung in der Mitte finden. Was ich sehr oft beobachte, ist, dass es sich manche Lehrende einfach machen und sich einen Anwendungsfall überlegen und diesen permanent weiter einsetzen. Das kann man machen, aber es wäre sinnvoll, wenn man das in der Retrospektive nochmal betrachtet und verbessert. Man kann schlecht von vornherein alles komplett richtig machen. Man muss eben jedes Jahr oder jedes Semester evaluieren und schauen, was könnte man anders machen und wie kann man das verbessern?

I: [14:58] Das heißt, die Ideen ist, sich einfach schrittweise ranzutasten?

B: [15:05] Genau das hatten wir auch mal in dem Modul Künstliche Intelligenz. Da haben wir drei Semester lang ein Framework entwickelt und immer weiter ausgebaut. Das ist machbar. Wenn man sich eine Aufgabe mitsamt einer Musterlösung nur einmal überlegt, dann macht das auch keinen Spaß.

I: [15:33] Jetzt geht es mal in die Richtung Gruppenarbeit oder auch die Motivation von Studierenden. Die erste Frage wäre, würdest du bei der Bearbeitung von Übungsaufgaben eher Gruppenarbeit oder Einzelarbeit bevorzugen, wenn man davon ausgeht, dass die Arbeit auch bewertet wird nachher?

B: [15:51] Es kommt drauf an. Man kann da jetzt nicht pauschal ja oder nein sagen. Also

ist es je nach Modul und je nach Aufgabe anders zu bewerten. Zum Beispiel beim Projektmanagement ist das Lernziel tatsächlich Teamarbeit auch zu fördern. Und die müssen das auch gemeinsam bearbeiten. Da würde ich jetzt auch nicht drauf verzichten wollen. Also bei den Programmieraufgaben, würde ich sagen, tendiere ich mittlerweile auch eher weniger zu Gruppenarbeit. Also zumindest mal am Anfang. Also vielleicht sollte eine Mischform da rauskommen, dass man sagt die Basics, die ersten Programmieraufgaben setzt jeder halt für sich allein um. Und dann gibt es vielleicht eine komplexere Aufgabe, wo man auch gemeinsam im Team eine Lösung entwickelt. Also man kennt das aus den Praktika, du kennst das wahrscheinlich auch, dass die Studenten versuchen, das irgendwie aufzuteilen. Einer macht Datenbanken, der andere Softwaretechnik und der dritte eben MCI. Und das ist nicht unser Ziel. Es soll jeder die Methodik kennenlernen und auch anwenden können.

I: [17:02] Und in Abgrenzung dazu nochmal existiert ja auch Pair-Programming. Wie würdest du das einordnen in Abgrenzung zur Gruppenarbeit?

B: [17:10] Das ist auf jeden Fall sinnvoll, aber vielleicht nicht am Anfang. In der jetzigen Zeit ist es schwierig, Studenten dazu zu bewegen, sich zusammenzusetzen. Wenn man das nicht wörtlich nimmt, dass man nebeneinandersitzt und einer programmiert, der andere zuschaut und kommentiert, kann man das sicherlich auch Remote machen. Dann braucht man eben entsprechende Tools dafür. Also das finde ich auf jeden Fall sinnvoll.

I: [17:58] Würdest du das dann der Gruppenarbeit vorziehen oder wären das nochmal zwei getrennte Anwendungsbereiche für dich?

B: [18:08] Das sind schon getrennte Bereiche. Und ja, ich würde jetzt eher eine Mischform bevorzugen. Zum einen sollte man auch alleine coden können. Man sollte aber auch die Vorteile von Pair-Programming kennenlernen. In größeren Projekten sind dann auch mal viele Entwickler involviert, mit dieser Situation sollte man auch Erfahrungen machen. Weil das kann man allein nicht lernen. Wenn man z.B. etwas eingecheckt hat und dann irgendwas kaputt geht, muss man schauen, wer dafür zuständig ist und so weiter. Solche Aspekte würde man allein nicht erfahren. Daher würde ich sagen, dass man das alles kombinieren sollte.

I: [19:15] Gehen wir mal in den Bereich Motivation über. Oft muss man Lernenden ein bisschen Druck machen, dass diese die Aufgaben überhaupt bearbeiten oder auch mit einer gewissen Motivation darangehen. Und da fallen mir zwei Ansätze ein. Einer ist eben

Druck durch Noten oder Individualisierung, also dass die Lernenden gar nicht die Möglichkeit haben, zu kopieren oder diese Lerngruppen zu bilden, von denen du gesprochen hast. Und die andere Möglichkeit wäre, dass man ein motiviertes Lernklima schaffen kann, sodass die Lernenden Lust auf die Aufgaben haben und vielleicht dieser Druck durch Noten oder Individualisierung gar nicht nötig ist. Die Frage ist, unter welchen Umständen meinst du ist welcher Ansatz der richtige oder kann überhaupt funktionieren?

B: [20:10] Also ich finde die Noten sowieso das Schlimmste, was es geben kann. Also ich behaupte mal, dass wir auch ohne Noten klarkommen würden. Für mich gibt es da nur zwei Seiten, entweder beherrscht jemand ein Werkzeug oder eine Methodik oder er beherrscht es nicht. Daher würde ich die Noten an sich, wenn ich es könnte, komplett abschaffen und am Ende nur einen Schein ausstellen, der bescheinigt, dass eine Methodik beherrscht wird oder nicht. Und wie diese Kompetenz zustande kommt oder wie lange man dafür braucht, spielt eigentlich gar keine Rolle. Die Anforderung ist also, dass ich etwas können muss und eine bestimmte Qualität liefern kann. Dies ist aber leider nicht möglich, obwohl es so viele wissenschaftliche Untersuchungen dazu gab, die besagen, dass es eigentlich nicht förderlich ist, durch Noten zu motivieren. Ich würde trotzdem versuchen, einen Kompromiss zu machen. Also es gibt sicherlich Studenten, die sich motivieren lassen. Das hängt auch von den Lehrenden ab. Also wie die Lehrenden ihr Modul gestalten. Es gibt so etwas wie agile Hochschuldidaktik. Das Buch kann ich nur empfehlen. Da geht es darum, dass man auch in der Lehre agil vorgeht und gemeinsam Ziele erarbeitet und der Lehrende dann in die Richtung der Ziele lenkt. Man kann dann den aktuellen Lernstand ermitteln und gemeinsam schauen, wie das Ziel erreicht werden kann. Das funktioniert bei einigen, wenn man das richtig umsetzt. Wenn das nicht funktioniert, dann muss man anders vorgehen. Es gibt immer noch Trittbrettfahrer, die einen Schubs brauchen. Wichtig ist meiner Meinung nach, klare Ziele zu formulieren. Was sind die Lernziele? Und wenn man die erreicht hat, dann kommt man auch weiter. Also unabhängig von Noten hat man z.B. Meilensteine und um jetzt weiter gehen zu können oder um zur Klausur zugelassen zu werden, muss man dann ein Praktikum bestehen. Da sind klare Qualitätskriterien definiert z.B. Definition of Done. Wenn einzelne Aufgaben erreicht werden, dann kommt man eben weiter. Wenn man das vollautomatisiert, dann muss man auch nicht wirklich in Kontakt mit Studierenden treten. Man muss das nur rechtzeitig kommunizieren und zeigen, wie das ablaufen kann. Dann würde man beispielsweise sehen, dass alle Tests auf grün sind, die Aufgabe somit erledigt ist und man kommt weiter. Es gibt auch Modelle, bei denen das Praktikum eine Teilnote ergibt und die Klausur dann als Individualleistung zählt. Dies ist zum Beispiel in unserem Modul Projektmanagement der Fall.

I: [23:29] Jetzt geht es etwas in Richtung Individualisierung. Was sind dir für Individualisierungsmethoden bekannt, die das Kopieren von Lösungen verhindern können?

B: [23:45] Die einfachste Lösung wäre, einfach mal die Testdaten zu individualisieren. Das würde schon mal dazu führen, dass man nicht so einfach kopieren kann. Die beste Lösung wäre natürlich, je nach Lernziel, wenn man unterschiedliche Aufgaben aus unterschiedlichen Domänen definieren würde. Dann müsste man sich quasi in die Domäne hineinversetzen, um die Aufgabe zu lösen. Da dies aber natürlich sehr aufwendig ist, glaube ich, dass die einfachste Variante wäre, individuelle Testdaten zur Verfügung zu stellen.

I: [24:48] Was wäre dann aus deiner Sicht wichtiger, dass man einen sehr hohen Grad an Individualisierung hat oder doch mehr Fokus legt auf die Geschichte bzw. Domäne, welche diese Aufgabe bestimmt?

B: [25:06] Es kommt darauf an, was man damit bezwecken will. Wenn man die Teamarbeit fördern will, dann macht es natürlich keinen Sinn unterschiedliche Domänen zu wählen. Wenn man jetzt einfaches Kopieren verhindern will, dann macht es schon Sinn, dass man dann mehr individualisiert. Aber ich kenne das auch aus eigener Erfahrung. Eigentlich sollte man tatsächlich mehr Pair-Programming fördern, sodass man sich gemeinsam Sachen anschaut, denn dabei lernt man viel mehr. Viele können das allerdings noch nicht, weil man das nie gelernt hat.

I: [25:59] Wie wäre denn deine Meinung zu abweichenden Schwierigkeitsgraden aufgrund von Individualisierung?

B: [27:34] Es gibt ja Vorgaben, dass Aufgaben, die benotet werden sollen, auch vergleichbar sein sollen. Daher muss man da schauen, dass es von der Komplexität relativ ähnlich ist.

I: [27:53] Wie viel Mehraufwand wärst du bereit zu investieren, wenn es darum geht, individualisierbare Aufgaben zu erstellen im Vergleich zu normalen Aufgaben?

B: [28:04] Ich scheue da keinen Aufwand. Je nachdem, wenn man jetzt ein Einzel Praktikum mit 300 Studenten betrachtet, wird das schon problematisch. Aber wenn man das automatisieren kann, dann ist das denke ich mal auch kein Problem.

I: [28:27] Das heißt, du würdest auf jeden Fall diesen Mehraufwand betreiben, wenn du

dadurch mit den positiven Aspekten von Individualisierung leben könntest?

B: [28:36] Auf jeden Fall. Ja.

I: [28:41] Du hattest eben auch automatische Überprüfung angesprochen. Und zwar, was muss deiner Meinung nach eine Übungsaufgabe für Eigenschaften aufweisen, damit man diese überhaupt automatisch überprüfen kann?

B: [28:59] Dann muss man schon gewisse Vorgaben machen können. Sei es z.B. Interfaces vorgeben, dass man dazu automatisierte Tests erstellen kann. Weil anders wird das nicht funktionieren, sonst müsste man das alles händisch prüfen. Es wäre auch möglich, dass die Studenten erstmal Test-Driven vorgehen und auch Tests implementieren. Das wäre auch eine Möglichkeit. Man könnte dann durch Individualisierung eine kleine Anzahl unterschiedlicher Aufgaben erstellen und dann jeweils ein Team Tests schreiben lassen und ein Team die Aufgabe auf Basis dieser Tests lösen lassen.

I: [30:07] Du sagtest eben, man müsste dann bestimmte Vorgaben machen, damit das automatische Überprüfen mittels Tests überhaupt geht. Das Problem dabei ist, dass dies die Freiheit der Studenten beim Lösen dieser Aufgabe einschränkt. Hast du Ideen, wie man da einen Mittelweg gehen könnte oder wie man das vereinen könnte, sodass die Aufgaben auf der einen Seite automatisch überprüfbar sind, und auf der anderen Seite Studenten aber noch Freiheiten haben?

B: [30:37] Da würde ich mich auf die Algorithmen zur Wegsuche Dijkstra und A* beziehen. Für die Algorithmen an sich kann man die wichtigsten Funktionen mithilfe von Pseudocode vorgeben. Diese könnte man somit auch automatisiert testen lassen. Das integrieren des Algorithmus in verschiedene individuelle Domänen würden wir dann den Studierenden überlassen und so Freiheit schaffen. Die Studierenden könnten dann auch für die jeweilige Domäne spezifische Anwendungstests schreiben. Das wäre ein Kompromiss, bei dem man nicht alles komplett vorgibt und die Studierenden im Design auch Freiheit haben.

I: [31:15] Dann geht es mal in Richtung Tools. Hast du schon mal Tools zur Aufgabenindividualisierung angewendet in der praxisorientierten Lehre? Und was für Erfahrungen hast du gemacht?

B: [31:29] Leider noch nicht, aber wir wären jetzt dabei. Aber das macht jetzt hauptsächlich einer meiner Kollegen und ich weiß gar nicht, wie weit er gekommen ist. In diesem

Semester bin ich tatsächlich nur mit dem Modul Projektmanagement beschäftigt. Wobei ich da auch langsam an dem Punkt angekommen bin, dass man das auch ziemlich gut unterstützen kann durch Tools. Also nicht nur im Bereich des Codings, sondern auch im Projektmanagement. Es gibt viele Lösungen zu Testdatenindividualisierung, welche man einsetzen kann. Wenn man konkret weiß, welche Testdaten man braucht, kann man sich zufällig genügend Datensätze generieren. Aber das ist jetzt nicht wirklich Individualisierung, sondern es handelt sich bloß um andere Testdaten.

I: [32:31] Was kennst du denn für Tools, mit dem man das auch in einem sehr großen Stil machen kann? Man hat ja oft mehrere hundert Studenten in einem Modul. Würde das in die Richtung gehen, was du gerade angesprochen hattest.

B: [32:42] Das weniger, also das sind eben nur unterschiedliche Testdaten, die man zufällig generieren kann. Aber das ist jetzt nicht wirklich die Lösung für das Problem. Da würde ich schon eher in die Richtung gehen, die ihr momentan einschlagt.

I: [33:03] Ein anderes Problem ist, dass wenn man dieses automatische Feedback von Tests betrachtet, ist das ja immer noch ein Unterschied zu dem Feedback von Betreuern, wenn diese eine Aufgabe korrigieren oder mit jemandem mal die Aufgabe durchgehen. In wie weit meinst du kann automatisch generiertes Feedback, das Feedback von Betreuern ersetzen?

B: [33:30] Meiner Meinung nach ist das schwierig, das würde man wahrscheinlich nicht so hinkriegen. Also man bekommt über automatische Tests schon konkretes Feedback, wenn man das Feedback vernünftig auf eine bestimmte Fehlermeldung reduziert. Aber man bekommt so kein Feedback zu dem eigentlichen Code. Es geht also um Feedback, welches auf Erfahrungen von Lehrenden basiert. Was man auch machen könnte, ist, dass man sowas wie ein Peer-Review Verfahren einbaut. Auch wenn alle Tests grün sind und alles wunderbar läuft, wird der Code nochmal durch ein anderes Team angeschaut, sodass es dann nochmal Feedback gibt. Als Lehrender würde man dann gewisse Qualitätskriterien vorgeben, wonach man den Code untersuchen sollte. Das wäre eine Möglichkeit und kann auch als Lernziel angesehen werden. Es gibt ja auch Tools, welche solche Verfahren unterstützen.

I: [35:45] Mit was für Plattformen hattest du bisher Kontakt, über die man generell Aufgaben an Studierende verteilen kann und auch später wieder Lösungen einsammeln kann?

B: [35:55] Ja, über Ilias, damit haben wir viel gemacht. Funktioniert tatsächlich auch ganz gut. Also wenn es jetzt nicht um Code geht, sondern um irgendwelche Abgaben. Dort gibt es Übungsmodule, wo man ziemlich feingranular einstellen kann, wann abgegeben werden soll, wer abgeben soll und was abgegeben werden soll. Für Source Code haben wir im Modul Algorithmik mal einen GitBucket verwendet. Da kann man dann nur ein Repository anlegen und schauen, dass die Studierenden das dann dort hochladen. Automatisierte Tests waren da aber auch nicht dabei. Früher haben wir die Aufgabenstellung an sich für Projekte im Master über Confluence und Jira verwaltet. Dass man das irgendwie automatisiert verteilt oder auch automatisiert Lösungen bekommt ist eben schwierig umzusetzen. Als ich hier anfing hatten wir damals eine Continuous Integration Lösung und Subversion eingesetzt. Da mussten damals schon die Studenten Maven nutzen und ihre Lösung commiten. Das hat nicht vielen gefallen, aber es wurde viel dabei gelernt.

I: [38:08] Eine Frage noch, und zwar Kommunikation zwischen den Lernenden und Lehrenden ist ja auch eine wichtige Komponente. Was ist deine Meinung zu modernen Messengern zur Kommunikation?

B: [38:26] Am besten sowas in Richtung Slack, dass man tatsächlich auch mal die Integration zu den Tools hat, aber dass man auch mitbekommt, wenn da irgendetwas gebaut wurde. Das ist etwas, was bei uns an der Hochschule wirklich fehlt. Momentan haben wir da Cisco Jabber, aber das ist eigentlich nur für die Lehrenden gedacht. Man kommt da an die Studenten nicht ran. Jetzt haben wir Zoom, darüber kann man auch kommunizieren. Es gibt einfach ziemlich viel an Tools und das wünsche ich mir unabhängig auch vom Coden, dass man irgendwie eine Lösung hätte für die Hochschule, welche auch Studierende mit einschließt. Diese kommunizieren nämlich ständig über Lösungen wie Discord, WhatsApp oder Facebook Messenger und müssen dann auch deren Daten freigeben. Dies muss nicht unbedingt sein, denn jeder hat einen Hochschul-Account und ich würde das auch trennen, auch aus Datenschutzgründen. Also meine Handynummer kennt auch schon fast jeder, aber ich wünschte mir, dass ich dann auch zuhause mal abschalten kann. Aber das fehlt wirklich noch bei uns. Ich würde mir das sehr wünschen, wenn es möglich wäre, dass man auch Integration über z.B. Chat Bots zu bestimmten Plattformen hat. Und dass man da vielleicht pro Modul auch separate Chats hat. Es gibt zwar das Ilias mit Forum. Aber dann bekommst du nicht direkt die Benachrichtigung. Das müsste man sich extra einschalten, dann bekommt man das per Mail. Da muss man sich dann einloggen und nochmal anschauen. Da ist natürlich Discord oder Slack ideal, aber leider noch nicht vorhanden für die Hochschule.

2.2.2 Experteninterview 2

Interviewpartner: Prof. Dr. Christian Kohls
Datum: 18.12.2020
Ort: Zoom-Meeting

00:00–50:27

I: [00:00] Zuerst einmal hätte ich eine sehr generelle Frage. Ein bisschen von oben drauf geschaut, was findest du sind die wichtigsten Werkzeuge, die man braucht, wenn man modernes Coding betreiben will?

B: [00:22] Natürlich auf alle Fälle eine integrierte Entwicklungsumgebung kennenlernen, im besten Fall auch mal zwei. Ich persönlich würde IntelliJ bevorzugen. Nicht nur, weil es Android unterstützt, sondern weil die von JetBrains einfach richtig gut sind. Eclipse ist auch ok, aber eben doch auch recht schwerfällig. Wenn man modern programmieren möchte, sollte man IntelliJ mal kennengelernt haben. Wenn man diese Entwicklungsumgebungen kennenlernt, muss man aber noch darauf eingehen, dass da ja viele Tools noch integriert sind, d. h. der Debugger wird von vielen Studierenden gar nicht mehr genutzt und auch da muss man als Student lernen, damit effektiv umzugehen. Auch die Building Tools sollte man verstehen. Und man muss eben auch verstehen, wie man visuelle Oberflächen baut, Libraries einbindet, diese ganzen Sachen muss man einmal gemacht haben. Dann natürlich Versionierung, dass man mit Git einmal gearbeitet hat. Dann sollte man im späteren Verlauf des Studiums auch Projektmanagement kennengelernt haben, ob das jetzt mit Trello, Miró oder Jira ist, spielt keine Rolle. Aber irgendwie müsste man es mal kennengelernt haben oder zumindest irgendwo einen Einblick in ein Backlog mit Features gehabt haben, die man abarbeiten muss. Und man sollte aus meiner Sicht auch einmal gelernt haben, wie man Klassendiagramme ordentlich skizziert. Ich finde Skizzen immer besser, als alles formal darzustellen, weil es andere formale Notationsformen gibt, die dafür besser geeignet sind. Aber die Strukturdiagramme, die muss man irgendwo zeichnen können, entweder mit Online Werkzeugen oder mit Visio, meinetwegen auch Powerpoint. Auch das geht, wenn ein Student keine Tools zur Verfügung hat. Aber man muss eben sowas kennengelernt haben.

I: [02:33] Du hattest eben Git erwähnt. Meinst du, es reicht, wenn man einfach sagt, benutzt Git mal über die IDE und die Magie dahinter sehen die Studenten dann gar nicht oder bist du der Meinung, man sollte vor allem am Anfang auch sagen, nimmt mal die Konsole und vielleicht können die Studenten dann die Funktionsweise auch besser durch-

dringen?

B: [02:53] Ja, ich glaube, man muss beides haben. Also vielleicht ist es zu Anfang ohne die Konsole gut. Ich habe früher die Konsole bei der Versionsverwaltung nicht so gerne gehabt, aber man muss das, glaube ich, einmal gesehen haben, um das zu verstehen. Ich habe in den ersten Semestern damals auch gezeigt, wie man mit Java per Hand eine Datei kompiliert, um die Magie dahinter zu sehen. Ich mache das aber inzwischen nicht mehr, weil es auch ein bisschen komplexer geworden ist. Also da lenkt man auch zu sehr ab, weil man mehr konfigurieren muss, wenn man per Hand kompiliert. Es kann so komplexer werden, als das, was eigentlich dahintersteckt. Also insofern ist es schwierig, da kann ich kein ja oder nein sagen. Pragmatisch mache ich es nicht, dass ich das zeige, weil ich denke, erst einmal geht es darum, überhaupt das Programmieren zu lernen. Und das ist schon Urwald genug. Und wenn man das draufhat, dann glaube ich, kann man sich auch so ein bisschen anschauen, was die Magie dahinter ist.

I: [04:04] Dann hätte ich mal eine ganz generelle Frage. Was macht für dich modernes Coding aus?

B: [04:16] Ich bin immer sehr orientiert an Entwurfsmustern und schaue, dass man von guten Praktiken anderer lernt. Das ist der eine Punkt, der wichtig ist. Der zweite Punkt ist, dass man eigentlich selbst dokumentierend programmiert. Man also gar nicht so viele Kommentare schreiben muss, sondern dass der Code so aufgebaut ist, dass eigentlich ganz klar ist, was damit gemeint ist. Dann ist für mich die lose Kopplung ganz wichtig, welche sich in vielen Entwurfsmustern widerspiegelt. Dass man also möglichst wenig Abhängigkeiten hat. Also dass es, wenn man ein System in seine Bestandteile zerlegt, möglichst wenig Abhängigkeiten gibt und dass man versteht, wie man ordentlich abstrahiert. Man fragt sich, wie kann ich mich denn von konkreten Erweiterungen unabhängig machen. Wie abstrahiere ich von Objekten auf Klassen ist das eine, aber das andere ist, wie abstrahiere ich auch von verschiedenen Implementierungen, sodass ich mich nicht mehr für die konkreten Implementierungen interessieren muss. Das muss man auf alle Fälle lernen. Dann ist Testen ein großes Thema, dass man also weiß, wie man testet. Man muss sich mit der Fehlerbehandlung mithilfe von Exceptions beschäftigen und dies auch verstehen. Wofür das ist und wie man das auch gut designt. Das man z.B. Nicht alles auffängt, sondern dass es manchmal auch gut ist, wenn ein Fehler geworfen wird. Code Sparsamkeit ist glaube ich eine ganz wichtige Sache. Einige Sprachen wie z.B. Kotlin unterstützen das. Es ist wirklich ein großer Unterschied zu anderen Sprachen, weil man viel schneller erkennt, was da eigentlich passiert. Aber auch wenn man dann Code schreibt. Ein klassisches Beispiel sind

Anweisungen wie z.b. if true return true else return false. Du glaubst gar nicht, was für umständlichen Code ich in Klausuren manchmal sehe. Also alles was richtig ist, bekommt volle Punktzahl. Aber ich habe teilweise jetzt Aufgaben drin, wo ich schlechten Code zeige und es Punkte dafür gibt, Eleganz einzubringen und zu vereinfachen. Ein Unterschied zur klassischen Objekt-Orientierung ist, dass dort immer ganz viel darauf geguckt wird, wie man Vererbungshierarchien aufbauen kann. Dort ist also Vererbung etwas Gutes. Und ich glaube, bei modernen Sprachen versucht man Vererbung zwar zuzulassen, aber zu vermeiden. Dass man da also ein bisschen mehr Bewusstsein hat, wo es Seiteneffekte geben kann. Und zwar nicht auf Ebene der Funktionen, sondern auf Klassenebene. Was auch noch wichtig ist, ist funktionale Programmierung. Vor 20 Jahren gab es das auch schon, aber hat noch nicht so eine große Rolle gespielt. Ich glaube, da kommt man heute einfach nicht drum herum, sich damit zu beschäftigen.

I: [07:58] Ja, du hast ja jetzt schon viele moderne Paradigmen genannt. Da wäre meine Frage, was fallen dir da für Programmiersprachen ein, die viele dieser modernen Paradigmen vereinen?

B: [08:09] Kotlin habe ich ja schon genannt. Also der Grund, warum wir auf Kotlin gewechselt sind, ist, dass eigentlich alle diese Paradigmen dort enthalten sind und dass Kotlin als Lehrsprache super ist, aber auch weil es industrienah ist. Scala ist eine Programmiersprache, die zwar ein bisschen akademischer ist, aber auch trotzdem sehr gut ist. Python halte ich auch für eine gute Sprache, um einige Konzepte darzustellen. Allerdings bin ich der Meinung, aber das ist wirklich so eine Weltanschauung, das typsichere Sprachen besser sind im Studium, auch, um Typen zu verstehen. Bis heute hat mich keiner überzeugen können, wofür man nicht typsichere Sprachen braucht. Ich glaube, eine interessante, aber auch gruselige Sprache ist JavaScript, weil sie viele Konzepte beinhaltet. An JavaScript kann man sich wirklich gut Konzepte anschauen, weil da z.B. auch funktionale Programmierung mit dabei ist. Also solche Konzepte sollte man kennen. Man muss den Unterschied zwischen funktionaler, objektorientierter und einfacher iterativen oder prozeduralen Programmierung kennen.

I: [09:38] Was würdest du in der Lehre höher priorisieren? Und zwar, wenn man den Fokus auf weit verbreitete Sprachen setzt oder eben auf modernere Sprachen, die vielleicht auch das Potenzial aufweisen, mal eine breite Anwendung zu finden? Du hattest ja auch Kotlin genannt, was ja schon in eine der beiden Richtungen geht. Aber was sagst du da so generell zu?

B: [09:59] Also man muss beides haben, man muss schauen, wie die Sprache verbreitet ist. Wie praxisrelevant ist die Sprache heute und wie praxisrelevant ist sie in fünf bis zehn Jahren. Und Kotlin ist da ein schönes Beispiel. Ich habe lange darüber nachgedacht, warum ich von Java auf Kotlin wechsle, weil einfach viel Java Code existiert. Aber ich habe gemerkt, dass man als Anfänger den Wald vor lauter Bäumen in Java Code nicht sieht. Und ich merke das wirklich beim Erklären. Es geht mit Kotlin viel besser. Also man lernt eigentlich mit Kotlin so zu programmieren, wie man in Java programmieren sollte, aber nie macht. Dass man mit Konstanten arbeitet, dass man Klassen final macht, nicht offen lässt und man keine RuntimeExceptions hat. Ich wäre nicht umgestiegen, wenn ich nicht wüsste, dass es ein Industriestandard ist. In dem Sinne, dass Google sagt, macht Kotlin und nichts anderes mehr. Kotlin ist also nicht eine empfohlene Sprache, sondern die empfohlene Sprache. Und ich glaube, selbst wenn man dann später nicht mit Kotlin entwickelt, lernt man damit richtig programmieren. Python wäre ein anderes Beispiel und ist insofern relevant, weil es im Bereich des Machine-Learning eine wichtige Sprache geworden ist. Ich behandle das ja im Modul Paradigmen der Programmierung, wo man mal andere Sprachen kennenlernt, aber nicht in der Tiefe durchdringt. Und das ist, glaube ich, auch wichtig, dass man ein bis zwei Sprachen richtig gut beherrscht, aber auch weiß, was es für andere Konzepte gibt. Also eine Sprache muss drei Faktoren gerecht werden. Die Sprache muss modern sein, nicht fancy modern, sondern eigentlich Konzepte aufgreifen, die etabliert sind. Sie muss zweitens eine Relevanz in der Industrie haben und sie muss drittens didaktisch geeignet sein. Wenn es alles zusammenfällt, dann landet man bei Kotlin. Aber das sind Überlegungen, die man an der Hochschule haben muss. Gerade Python und JavaScript sind auch tolle Sprachen und die sind super Industrie relevant, aber sie verführen auch zu schlechtem Code. Gerade JavaScript.

I: [12:30] Du hast dich eben schon mal für Clean Code ausgesprochen. Meine Frage ist, wie kann man Lernende dazu bringen, dass sie sich auch an bestimmte Richtlinien halten, welche zu sauberem Code führen?

B: [12:46] Im Idealfall machen sie das ja von vornherein. Aber ich glaube, man muss auch ein bisschen in die Probleme reinlaufen, um zu verstehen, warum sauberer Code wichtig ist. Es gibt so ein didaktisches Muster, das heißt “Feel the Pain“. Also man muss schon einmal gefühlt haben, warum komplexe Systeme zusammenkrachen, wenn man zu viel Abhängigkeiten hat. Dann kann man verstehen, warum ich mit Entwurfsmustern anfange. Wenn ich mit diesen Konzepten gleich zu Anfang anfange, dann verstehen die meisten nur Bahnhof. Ja, also das Einführen von Interfaces ist schon so eine Sache, wo viele fragen, warum mache ich denn das eigentlich? Da muss man viel Zeit für darauf verwenden, um

das zu erklären, warum man diese Konzepte benötigt, welche ja alle erst Relevanz haben, und das ist das Gemeine, wenn ich komplexe Systeme bauen soll. In der Lehre habe ich aber typischerweise keine komplexen Systeme. Das heißt, ich muss eigentlich auch Aufgaben designen, wo Studierende den Code verbessern können. Und da muss man natürlich auch beim Code Review so Basic Sachen machen. Das fängt an bei Code Konventionen, dass man eben Klassennamen großschreibt und nicht klein oder wo man Sachen kürzer schreiben kann. Ich glaube, das kann man wirklich nur individuell mit Feedback machen. Das kann man nicht im Frontalunterricht vermitteln, das muss man korrigieren, bei dem, was die Studierende machen.

I: [14:22] Gibt es denn da für dich praktikable Wege, wie man diese Richtlinien auch erzwingen kann? Also dass man das nicht erst im Feedback sagt, sondern schon bei Übungsaufgaben sauberen Code ohne manuelle Korrekturen erzwingt?

B: [14:38] Du meinst, dass es automatisiert wird?

I: [14:39] Ja.

B: [14:41] Du kannst natürlich vorgeben, wie viele Klassen gebaut werden müssen, also wie die Vererbungsstruktur ist. Du kannst Vorgaben machen, dass eine Klasse final sein soll. Man kann eine Vorgabe machen: Setzen Sie alles, was nicht veränderbar oder zugreifbar sein soll, auf private. Bei Aufgabenstellungen von uns sind das Anforderungen. Also was ist die Vererbungshierarchie? Was soll private sein? Welche Eigenschaften sollen berechnet sein oder welche Eigenschaften sollen zu Anfang schon gesetzt werden? Also da kann man einfach Requirements definieren. Dass kommentiert werden muss und Konventionen eingehalten werden, ist auch klar. Was ich bisher nicht gemacht habe, ist zu sagen, ihr müsst in einer geringen Anzahl von Lines of Code zurechtkommen, könnte man ja auch machen. Es dürften dann z.B. nur zehn Zeilen Code sein und dann gibt es eigentlich nur eine bestimmte Form, wie man es lösen kann. Eine Sache, die man machen kann, um das auch zu vergleichen, ist, dass man sagt, implementiert denselben Algorithmus auf unterschiedliche Weise. Sodass man auch die Vor- und Nachteile abwägen kann. Also z.B. einmal rekursiv und einmal iterativ. Einmal zeigen, was passiert, wenn ich eine Funktion habe, die zwei Objekte übergeben bekommt, im Vergleich zu, ich habe ein Objekt, wo es eine Methode gibt, wo ein weiteres Objekt übergeben wird. Man muss also die gleiche Methode zweimal oder mehrfach zeigen, um die Unterschiede zu sehen. Ich will einfach zeigen, welche Möglichkeiten gibt es eigentlich? Und wie sind da die Zusammenhänge? Ein Beispiel wäre eine Liste, dessen Größe ich berechnen möchte. Speichere ich das in einer Ei-

genschaft, habe dann zur Laufzeit aber Seiteneffekte oder berechne ich das jedes Mal? Und das miteinander zu vergleichen, welche Vor- und Nachteile das hat, ist auch ein guter Weg.

I: [17:06] Du hast ja auch schon einige Richtlinien genannt, die man Clean Code zuordnen kann. Was wären denn da besonders Wichtige, die man unbedingt überprüfen sollte?

B: [17:33] Ich muss noch mal ganz kurz auf Kotlin zurückkommen. Einer der Gründe, warum ich Kotlin damals eingeführt habe, ist, dass Kotlin als Sprache dazu führt, dass ich saubereren Code schreibe. Denn Kotlin zwingt mich zu bestimmten Sachen, welche ich bei anderen Sprachen überprüfen müsste. Also ich kann z.B. keine NullPointer verwenden, außer ich mache es explizit und prüfe dann, ob der Wert nicht vielleicht null ist. Ja, das ist ein häufiger Fehler, dass Variablen nicht initialisiert werden oder vergessen werden. Der nächste Punkt ist, dass es auch häufig Seiteneffekte gibt. Also man muss ganz stark gucken, dass keine Seiteneffekte auftauchen, weil man von beliebigen Methoden auf irgendwelche globale Variablen zugreift. Und wenn man aus dem Modul AP1 noch weiß, dass globale Variablen nicht gut sind, nutzt man sie spätestens in dem Modul AP2 dann doch wieder. Die Studenten meinen dann, nur weil es Variablen innerhalb irgendwelcher Objekte sind, auf die jeder zugreifen kann, ist es dann plötzlich nicht mehr global. Aber ich muss es eben in einem Objekt oder in einer Klasse schützen und diese Zugriffe kapseln, was eigentlich ganz wichtig ist. Also wenn ich jetzt von einer Objektorientierung spreche, muss man schauen, ob ordentlich abstrahiert ist. Also ob ich verstehen kann, was die wesentlichen Eigenschaften sind. Da muss ich gucken, wie ich komplexe Objekte zusammensetze und auch in einer Klasse das zusammenfasse, was zusammengehört. Also, dass ein Element richtig auseinandergenommen wird. Bei einem Auto z.B. nimmt man nicht den halben Reifen weg, sondern betrachtet den ganzen Reifen als einen Teilnehmer. Die Kapselung als Konzept zu verstehen ist glaube ich ganz wichtig. Und dann zu schauen, wofür man die Unterscheidung zwischen Schnittstellen und Umsetzungen braucht. Entwurfsmuster finde ich immer ganz gut geeignet, weil diese viele von den Clean Code Prinzipien integrieren. Ein ordentliches Muster ist ja gerade eines, welches einen sauberen Code abbildet. Da kann man ganz gut dran erklären, was die Designentscheidungen sind, die man gegeneinander abwägt. Wenn man z.B. in einer bestimmten Situation bestimmte Anforderungen und Einflussfaktoren hat, dann kann ich diese mithilfe von Mustern unter einen Hut bringen, muss dann aber mit den Konsequenzen leben. Das Abwägen ist ganz wichtig und sollte man trainieren. So kann man wirklich versuchen, eine Balance herzustellen. Das ist, glaube ich, ein ganz wichtiger Punkt.

I: [20:51] Ich hätte da noch eine Frage zu Domain Driven Design. Und zwar ist DDD

ja ein Ansatz, mit dem man sehr komplexe Software modellieren möchte und dabei sehr fachlich vorgeht. Wenn man das allerdings lehren möchte, hat man dann das Problem, dass Übungsaufgaben ja eine gewisse Beschränktheit haben, weil die können ja nicht so komplex sein. Und die Frage ist, was muss man da beachten, wenn man Übungsaufgaben erstellt, die solche Ansätze lehren sollen?

B: [21:28] Also das ist ja insofern auch interessant, weil man ja auf einer ganz anderen Ebene ist am Anfang der Ausbildung. Man möchte ja trainieren, dass man etwas anhand einer Spezifikation umsetzen kann. Da brauche ich dann aber noch kein DDD, weil ich dann schon Vorgaben habe. Was dann ja spannend ist, ist die Domäne zu modellieren und zu schauen, was ich für Events beachten muss oder welche Datenmodelle ich brauche. Und ich glaube, das ist etwas, was man anhand einfacher Domänen erproben kann, mit denen die Studierenden eigentlich schon vertraut sind. Die Frage ist auch, welchen Aspekt von DDD man eigentlich vermitteln will. Also dass ich entweder in eine Domäne gehe, wo ich eben kein Domänenexperte bin und erst verstehen muss, wie die Domäne funktioniert und deshalb auch Methoden kennen muss, mit denen ich das Wissen aus anderen herauskitzeln kann. Und andersherum glaube ich aber, dass es auch hilfreich ist, sich die Methoden anzuschauen, die man braucht, wenn man eigentlich selber schon Experte ist und trotzdem immer schaut, wie kann ich z.B. ein ordentliches Glossar aufbauen. Ich habe das bisher noch nicht gemacht, muss ich gestehen. Ich weiß ja, dass der Erstbetreuer von der Arbeit viel mit DDD macht und ich war tatsächlich letztes Jahr auch auf der DDD Konferenz. Ich bin am überlegen, ob ich das nicht bei Social Computing mit einbinde. Also das ist ein Kurs, den ich gebe, wo es eigentlich um soziotechnische Systeme geht. Da könnte man, obwohl es nicht ums Programmieren geht, trotzdem DDD mal anwenden und ausprobieren. DDD ist natürlich auch für den Bachelor schon sehr ambitioniert, das wäre im Master ein bisschen einfacher.

I: [25:07] Und dann hätte ich mal ein paar Fragen konkret zur Organisation von einem Praktikum oder solchen Aufgaben. Und zwar würdest du da eher Programmieraufgaben in Gruppenarbeit bevorzugen oder Einzelarbeit, wenn es darum geht, diese Aufgaben auch zu bewerten im Nachhinein?

B: [25:28] Eigentlich eine Kombination, so haben wir das auch gemacht. Also erstmal sind Gruppen sehr unterschiedlich. Es gibt die Gruppe, wo nur einer was macht und der andere schaut zu. Das ist für den, der schwächer ist, wirklich ein Lernhindernis, weil es bequem ist und der, der gut ist, will ja was machen. Der hat dann keine Geduld und macht dann ganz schnell. Das ist der negative Fall vom gemeinsamen Arbeiten. Der po-

sitive Fall ist natürlich, dass beide gerne wirklich programmieren wollen. Der eine kann es z.B. schon und erklärt es dem anderen und wird durch das Erklären auch nochmal ein Level hochgehoben, andersherum genauso. Insofern war da mein Modell zu sagen, ihr dürft zwar in Gruppen entwickeln, aber abgeben und erklären muss jeder einzeln. So kann man sicherstellen, dass wirklich jedes Gruppenmitglied auch wirklich alles kann. Ab dem dritten Semester lasse ich dann sogar größere Gruppen zu. Da muss dann jeder nur ein kleines Stückchen erklären und dann bin ich auch ein bisschen toleranter, wenn es nicht jedes Gruppenmitglied kann, weil das dann deren Verantwortung ist. Die ersten beiden Semester bin ich noch bereit, streng zu sein und danach muss es eigenverantwortlich sein und zur Not erst in der Prüfung knallen.

I: [27:18] Dazu hätte ich nochmal eine Frage. Wie würdest du im Vergleich dazu Pair-Programming einordnen? Also nochmal in Abgrenzung zur Gruppenarbeit?

B: [27:27] Also bei Pair-Programming können es ja eigentlich beide, deswegen ist es das bessere Modell als wirklich größere Gruppenarbeiten. Also gerade nachher dann bei Gruppenarbeit in höheren Semestern geht es ja auch darum, dass man Aufgaben verteilt. Und bei Pair-Programming stelle ich sicher, dass wirklich beide jede Zeile Code gemeinsam angefasst haben. Und wenn man das tatsächlich etablieren könnte, dass die Studenten sich auch abwechseln und jeder mal die Tastatur in der Hand hat, halte ich das für eine sehr gute Möglichkeit. Leider ist das aber nicht das, was ich beobachte im Praktikum. Wenn ich da die Gruppen sehe, dann ist derjenige, der schneller tippen kann, an der Tastatur. Also an sich finde ich das Konzept supergut, aber in dem Bewusstsein, dass das von den Studierenden so umgesetzt wird, wie man sich das eigentlich wünschen würde. Man kann sie ja auch nicht dazu zwingen, man kann sie ja nur ermuntern.

I: [28:32] Thema Motivation: Man muss den Studierenden auch oft so ein bisschen Druck machen, dass sie die Aufgaben überhaupt bearbeiten oder mit einer bestimmten Motivation bearbeiten. Und da gibt es ja verschiedene Arten, wie man diesen Druck generieren kann. Also einmal kann man natürlich Noten einführen oder Aufgabenindividualisierung schaffen, wo das Kopieren dann gar nicht mehr möglich ist. Oder man hat irgendwie ein sehr motivierendes Lernklima, wo die Leute gar nicht das Bedürfnis haben, abzuschreiben, zu kopieren und die Aufgabe somit selber lösen möchten. Die Frage ist, unter welchen Umständen meinst du, funktioniert überhaupt welcher Ansatz oder sollte auch bevorzugt werden?

B: [29:13] Also ich kann ja mal erzählen, wie ich das gemacht habe und was ich vielleicht

anders machen möchte. Also bisher war das so: Ich habe tatsächlich als extrinsische Motivation Bonuspunkte vergeben für diejenigen, die ein bisschen mehr gemacht haben. Oder ansonsten mussten bei der Abnahme Live-Aufgaben gelöst werden. Ob Code kopiert ist, darauf habe ich aber eigentlich nicht geachtet. Also die Ansage war eigentlich, ihr dürft kopieren. Also das überprüfen wir nicht, empfehlen tun wir das aber nicht. Das ist eine sinnlose Diskussion zu sagen: Ist das jetzt kopiert oder nicht kopiert? Weil man muss den Studenten schon klarmachen, wenn ihr da Code kopiert, betrügt ja nicht mich als Dozent, sondern ihr betrügt euch ja selber. Ihr seid ja hier, um zu lernen, ihr habt euch ja selber dafür entschieden, herzukommen und programmieren zu lernen. Und wenn ihr unsere Hilfe nicht in Anspruch nehmt, dann habe ich das so zu akzeptieren. Dann müsst ihr euch überlegen, ob ihr hier richtig seid oder nicht. Und diese Diskussion, kopiert oder nicht kopiert, die führen wir gar nicht. Das Einzige, was wir machen, ist zu sagen, hier habt ihr eine kleine Aufgabe, ändert mal den Code. Wenn ihr das nicht könnt, dann gibt es da keine Punkte für. Also damit ist diese Aufgabe eben nicht bestanden, weil wir sehen können, das hat nicht funktioniert. Ob ich Bonuspunkte vergebe, weiß ich noch nicht. Ich möchte das Modell ein wenig umbauen. Bisher war das so, dass ein Praktikum aus mehreren Aufgaben besteht, meistens so drei bis vier im Laufe des Semesters. Wenn du also bei der dritten oder vierten Praktikums Aufgabe nicht mehr dabei bist und es nicht mehr schaffst, dann ist das gesamte Praktikum nicht bestanden. Was dazu führt, dass da, wo es eigentlich schwieriger wird, also da, wo man eigentlich hinkommen will, wir mehr Augen zudrücken. Ja, denn die sind ja dann schon bis zur dritten oder vierten Aufgabe gekommen und dann will man sie nicht durchfallen lassen, weil da müssen sie ein Jahr warten. So die Idee, die wir alle haben, ist so ein bisschen Gamification orientiert. Wenn ich z.B. ein Spiel habe und ich ein Level nicht schaffe, dann musstest du früher auch bei den alten Arcade Spielen wieder von vorne anfangen. Das ist echt frustrierend. Wenn du dir heute Spiele anschaust und ein Level nicht schaffst, dann kannst du auf dem gleichen Level weitermachen. Und das ist eigentlich die Idee, zu sagen, wir haben vier Aufgaben, die vier Aufgaben sind vier Level. Wenn du Aufgabe eins bzw. Level eins nicht geschafft hast, musst du nochmal Level eins machen und das kann man öfter anbieten, also mehrfach im Semester, alle vier Level. Und wenn man dann Level zwei geschafft hat, geht man ins Level drei. Wenn man Level drei nicht schafft, falle ich nicht wieder auf null zurück, sondern Level eins und Level zwei bleiben geschafft. Damit wollen wir motivieren und betonen, dass schon zwei Level geschafft wurden. Wir vermitteln dann, dass die Studenten dann schon so weit gekommen sind und das dritte Level fast geschafft haben und es mit ein bisschen Zeit auch schaffen können. Wie wir das vom Zeitrahmen her organisieren, ob wir eine Woche Level eins machen, dann Level zwei in den nächsten Wochen, dann Level drei oder ob wir sagen: Jeden Montag ist Level eins, jeden Dienstag ist Level zwei und

so weiter. So kann man, wenn man will, in einer Woche oder über einen längeren Zeitraum das Praktikum bestehen. Das müssen wir nochmal überlegen. Die Idee dahin ist, das ganzjährig anzubieten und den Stress rauszunehmen. Momentan ist leider folgende Denkweise sehr präsent: Wenn ich jetzt diese Aufgabe nicht abgenommen bekomme, dann habe ich das Praktikum nicht bestanden und dann fange ich von vorne an. Es geht also nur um dieses Bestehen, Bestehen, Bestehen. Und bei dem anderen Modell würde man halt versuchen zu sagen: Erreicht, erreicht, erreicht. Man erreicht also immer mehr, das wird dir nicht weggenommen und dann kann man daran arbeiten, aufs nächste Level zu kommen. Da versuchen wir die intrinsische Motivation mehr anzusprechen.

I: [33:55] Wie du schon sagtest, das geht jetzt in Richtung Gamification, Motivation von den Studierenden. Ich würde jetzt aber trotzdem mal ein paar Fragen zur Individualisierung stellen. Also wenn man wirklich verhindern will, dass Leute kopieren, was man da machen kann. Und da würde ich erst einmal fragen: Was kennst du da für Individualisierungsmethoden, welche wirklich dafür sorgen, dass einfaches Kopieren von Lösungen nicht möglich ist oder eben erschwert wird?

B: [34:20] Also das, was wir gemacht haben, ist zu sagen: Du kannst deinen Code meinetwegen sogar kopieren. Aber es gibt eine live Aufgabe und von den Live Aufgaben haben wir ganz viele und es wird zufällig eine ausgewählt.

I: [34:31] Also quasi ein Aufgabenpool?

B: [34:33] Genau und durch die elektronischen Prüfungen jetzt in Corona-Zeiten musste ich Pools an Fragen aufbauen für die Prüfung. Das geht eigentlich ganz gut. Das kann ich mir auch in Zukunft für die Praktika vorstellen. Wenn man dieses Gamification Modell hat, dann gehst du in Level eins rein, startest Ilias und dir wird eine von 150 Aufgaben zugeteilt. Und wenn jetzt jemand sagt, er sei so gut, dass er sich 150 Lösungen kopiert und das organisieren kann, ist das ja auch eine Leistung. Aber ich glaube, das bekommt man mit dieser Randomisierung ganz gut hin. Und wenn man dann nochmal randomisiert und kombiniert 150 Aufgaben mit 150 anderen Themen, dann hast du 150 mal 150 Aufgaben Typen. Das kann kein Mensch kopieren, das Ziel wäre dann, sehr individualisierte Aufgaben zu basteln.

I: [35:42] Ich hätte dazu nochmal eine Frage: Was findest du da wichtiger, dass man einen sehr hohen Grad an Individualisierung hat, was das Kopieren immer schwerer macht, oder sollte man doch mehr Fokus legen auf die Geschichte, welche den Aufgabenkontext be-

stimmt?

B: [35:59] Ja, beides muss wichtig sein. Das ist tatsächlich die Herausforderung. Bisher habe ich das so gehabt, dass Aufgabe eins bis vier aufeinander aufbauen. Die Aufgabe wird immer komplexer. Und das macht das ganze natürlich schwieriger. Und da weiß ich noch nicht, wie man das abbilden kann, dass man beides hat, also randomisierte Aufgaben und aufeinander aufbauende Aufgaben. Und ich glaube schon, dass wichtig ist, dass der Schwierigkeitsgrad immer größer wird, aber das das bisher Erreichte irgendwie weiter verwendet wird. Also ist die Struktur der Aufgabe auch superwichtig. Und die sind eigentlich auch immer gleich. Also ich habe in den ganzen letzten Jahren immer die gleichen Aufgaben abgefragt mit verschiedenen Aufgaben Typen. Also man muss irgendwo immer Objekte miteinander verbinden, verknüpfen, erzeugen, durchlaufen, irgendeine Vererbung mit drin haben, irgendwo eine Methode überschreiben. Also die Sachen sind tatsächlich auf einer abstrakteren Ebene immer dieselben, die man in vielen Variationen durchlaufen kann.

I: [37:12] Wie wäre denn da deine Meinung zu abweichenden Schwierigkeitsgraden? Wenn du sagst, du hast ja ein Pool aus Aufgaben, können die ja teilweise sehr stark variieren.

B: [37:27] Das wird man nie 100 prozentig genau hinkriegen. Die Problematik hatte ich aber in den elektronischen Prüfungen noch viel stärker, weil wenn man da eine Varianz der Aufgabe hat und der Schwierigkeitsgrad sehr voneinander abweicht, dann wäre es wirklich unfair, wenn man dieselbe Prüfung betrachtet. Also man muss ganz stark darauf aufpassen, dass die ungefähr gleich vom Umfang her sind. Man wird sie nicht eindeutig gleich hinbekommen. Da muss man mit leben, das ist einer von den Nachteilen. Aber vielleicht der kleinere Nachteil.

I: [38:00] Das ist natürlich auch immer mit einem gewissen Mehraufwand verbunden, wenn man diese Art von Aufgaben erstellt. Wie viel Mehraufwand wärst du da bereit zu investieren für den Zweck der Individualisierung?

B: [38:14] Dadurch, dass ich denke, dass man das einmalig aufbauen muss und das dann richtig gut funktioniert, habe ich eigentlich für das nächste Semester geplant, auch ein bisschen Zeit reinzustecken. Und die andere Sache ist ja auch, dass man einen Pool ja auch stückweise aufbauen kann. Man muss ja nicht sofort den gesamten Pool haben, weil die Studenten beim ersten Durchlauf z.B. noch keine alten Lösungen haben. Wenn man den Pool aber erweitert, dann hat man nachher irgendwann ganz viele.

I: [38:46] Jetzt würde ich mal ein bisschen Richtung Automatisierung übergehen. Was müsste für dich eine Übungsaufgabe für Eigenschaften aufweisen, damit man diese auch automatisch prüfen kann?

B: [39:13] Also damit man automatisch prüfen kann, muss natürlich das Ergebnis eindeutig prüfbar sein. Das ist die Herausforderung. Aber mich interessiert ja nicht nur der Output oder die Objekt Struktur, sondern ich möchte ja teilweise auch prüfen, ob die richtigen Variablen angelegt sind oder ob die eigentlich Klassenstruktur richtig ist. Also wenn ich z.B. fünf mal eins rechne, dann ist fünf das richtige Ergebnis, aber ich möchte keinen Algorithmus haben, der fünf Mal um eins inkrementiert. Und so etwas müsste man irgendwo auch überprüfen können. Ich muss mir also vorher überlegen, was könnten typische Fehlkonzepte sein. Also wenn ich z.B. eine Liste von Zahlen habe und es sollte der Durchschnittswert berechnet werden, dann muss man sich überlegen, was da passieren kann. Vielleicht wird dann vergessen, durch die Anzahl der Elemente zu teilen. So kann man besseres Feedback geben, als einfach zu sagen, dass etwas nicht stimmt. Oder man prüft z.B. ob das letzte Element nicht mitgezählt worden ist. Man muss einfach auch aus den bisherigen Prüfungen überlegen, was typischerweise falsch gelaufen ist und das in der Automatisierung abbilden.

I: [40:57] Du sagtest eben, dass man bestimmte Vorgaben macht. Allerdings nimmt das den Leuten auch eine gewisse Freiheit weg in der Aufgabenbearbeitung. Die Frage lautet: Wie kann man denn gewährleisten, dass Aufgaben sowohl automatisch überprüfbar sind, aber auch die Studenten trotzdem Freiheiten haben in der Aufgabenbearbeitung?

B: [41:40] Also im Prinzip hat man bei der automatisierten Prüfung ja immer ein bisschen Freiheit, weil man ja nicht alles automatisiert prüfen kann, es sei denn natürlich man parst den Code Zeile für Zeile. Ich glaube, es gibt zwei Ebenen. Man kann natürlich schon automatisiert testen, was allerdings nicht mehr ganz so gut funktioniert, ist eben das Feedback, wie man den Code verbessern kann. Also was man natürlich schon auch automatisiert machen kann, ist zu sagen: Warum ist das nicht private oder warum ist diese Klasse nicht final, wenn es keine Ableitungen gibt? Vieles davon machen auch gute Entwicklungsumgebungen schon, geben Warnhinweise, die man berücksichtigen sollte und am besten mit einbezieht. Man könnte z.B. sagen: Ihr dürft machen, was ihr wollt, aber verwendet nichts, was deprecated ist. Oder man versucht auch automatisiert zu schauen, ob man auf interne Eigenschaften einer Klasse zugreifen kann. Man könnte ja auch so eine Art Attacke machen: Dein Ergebnis stimmt nicht, warum nicht? Ja, weil ich habe Zugriff

darauf gehabt und ich habe einfach mal eine fünf eingesetzt. Aber das wird natürlich schon komplexer.

I: [43:21] Ich würde dann mal in Richtung Tools übergehen, mit denen einiges, was wir jetzt auch so besprochen haben, umsetzbar ist oder diese Tools zumindest unterstützend wirken. Hast du schon mal Tools zur Aufgabenindividualisierung genutzt? Was für Erfahrungen hast du damit gemacht? Du sagtest ja eben, du hast das mit Ilias und Pools mal ausprobiert. Hast du da auch noch was anderes ausprobiert?

B: [43:43] Ich habe tatsächlich dafür nur Ilias verwendet und bei einer anderen Möglichkeit, die ich mal angewendet habe, mussten sich die Lernenden selber eine Aufgabe individualisieren. Da hatte ich einfach eine Seite mit 1000 Themen und das Thema eines jeden einzelnen hat sich einfach aus den letzten drei Ziffern der Matrikelnummer ergeben. Zum Beispiel baue eine Hierarchie aus drei Klassen, eine Oberklasse, zwei Unterklassen zum Thema Fischerei oder zum Thema Politik. Und auf welcher Ebene das Thema interpretiert wird, ist mir dann eigentlich relativ egal, aber Hauptsache es passt irgendwie zu dem Themenfeld. Jemand, der das Thema Hotel hat, kann dann relativ schlecht von jemand kopieren, der das Thema Politik hat. Und das war so eine Möglichkeit, wie man auch relativ einfach automatisieren kann.

I: [44:53] Kennst du da noch zusätzliche Tools, mit denen man das auch im großen Stil betreiben kann? Wenn man jetzt sagt, man hat irgendwie 200 Studierende und will für diese Aufgaben individualisieren. Das kannst du mit dem Pool natürlich auch machen und auch jetzt mit dieser zweiten Variante, aber gibt es da noch etwas, was dir einfällt?

B: [45:09] Leider nicht, aber ich habe da auch nicht gezielt recherchiert.

I: [45:16] Ich würde dann mal zu anderem Tooling übergehen, und zwar hatten wir ja eben über automatische Aufgabenüberprüfung geredet. Und daraus resultiert natürlich ein anderes Feedback, wie das Feedback von Betreuern, welche auch Aufgaben manuell korrigieren und auch nochmal mit Studenten Gespräche führen. Inwiefern meinst du, kann automatisches Feedback das Feedback von Betreuern ersetzen?

B: [45:42] Also ich glaube, es handelt sich dabei um eine Art First Level Support. Ich glaube, wenn etwas funktioniert, ist ja erstmal alles gut. Auch da, wo es Fehler gibt, kann ganz viel automatisiert werden. Dann hat man tatsächlich auch mehr Zeit, sich auch auf einer höheren Ebene mit Problemen zu beschäftigen. Das sind einfach verschiedene Le-

vel von Support. Also wenn man sagt, man fängt vielleicht mit der Automatisierung an, und dann kann man einen Betreuer fragen. Und ich glaube durchaus, dass einige Sachen auch ganz ohne Betreuer abgebildet werden könnten. Die wertvolle Ressource Zeit, die man miteinander hat, kann ich vielleicht eher damit verbringen, nochmal zu erklären, wie Klassenstrukturen aufgebaut sind, statt nochmal zu erklären, wie man einen Durchschnittswert berechnet.

I: [46:46] Ich würde jetzt mal zur nächsten Frage übergehen. Jetzt geht es mir um die automatische Verteilung von Aufgaben und das Abgeben von Lösungen. Was hast du da für Plattformen genutzt, um dies zu gewährleisten, vielleicht auch neben Ilias?

B: [47:08] Da habe ich neben Ilias leider noch keine anderen genutzt.

I: [47:31] Kennst du denn in dem Bereich noch andere oder hast du welche im Visier, die du interessant findest?

B: [47:45] Also was ich interessant finde, ist der ganze Bereich der Kotlin Koans. Also da habe ich auch gerade eine Bachelorarbeit zu am Laufen. Das sind genau solche Programmier Challenges, bei denen man über Tests festlegt, ob die Aufgabe richtig gelöst ist oder nicht. Und man kann eben Feedback geben, warum etwas richtig gelöst ist oder nicht.

I: [48:22] Werde ich mir auf jeden Fall mal anschauen. Ich hätte noch eine letzte Frage zu Feedback und Beratung. Was hältst du von modernen Messengern als Kommunikationsplattform zum Austausch zwischen Lehrenden und Lernenden.

B: [48:39] Ja, finde ich supergut. Aber ich kenne noch keine, die das erfüllen, was man eigentlich braucht. Nämlich Gruppen bilden und sagen können, dass man mal nicht für einen Studenten verfügbar ist. Denn ich sag mal so, mit WhatsApp habe ich ein Datenschutzproblem. Ich mache mit allen gerne WhatsApp, aber nicht mit Studierenden. Ich möchte da in beide Richtungen keine Mobilfunknummern austauschen. Ich möchte auch niemanden ausschließen, der nicht WhatsApp nutzt. Also Skype ist auch so ein schönes Beispiel, wenn dann Skype bei mir lief, dann bekommt man auch mal schnell eine Nachricht von einem Studenten. Und du hast bei Instant Messaging immer die Erwartungshaltung, dass du sofort antwortest. Das geht aber nicht. Bei Freunden oder dem Arbeitsteam gerne, aber nicht bei Studenten. Das geht also nach hinten und wird dann unabsichtlich ignoriert. Anders als bei einer E-Mail, die ich irgendwann abarbeite. Und das ist glaube ich die Herausforderung, dass man sich überlegt, wie kann eine gute Platt-

form aussehen. Also Discord ist zum Beispiel ein gutes Tool, das wäre von der Art und Weise her eine gute Möglichkeit, mit Channels zu arbeiten und so weiter. Ist aber auch noch nicht unbedingt etwas, was man in der Lehre einsetzen kann, auch aus Datenschutzgründen. Ich kann ja niemanden dazu zwingen. Aber prinzipiell finde ich so Messenger Lösungen gut, da würde ich mir nur eine Art Status wünschen, durch den ich zeigen kann, dass ich erreichbar bin oder nicht. Oder dass sogar Nachrichten gesammelt werden, die ich dann später abarbeiten kann.

2.2.3 Experteninterview 3

Interviewpartner: Prof. Dr. Stefan Bente
Datum: 29.12.2020
Ort: Zoom-Meeting

00:00–51:51

I: [00:01] Was sind deiner Meinung nach die wichtigsten Werkzeuge, die man braucht, wenn man modernes Coding betreibt?

B: [00:16] Also du musst vertraut sein mit einer modernen IDE, also üblicherweise IntelliJ oder VS Code. Du musst vertraut sein mit Git, mit den wichtigsten Kommandos zumindest. Wir haben das im Kollegenkreis mal diskutiert und ich habe mich da überzeugen lassen, dass die Verwendung von der Git Bash eine gute Sache ist, weil man dadurch nochmal ein bisschen besser versteht, wie Git eigentlich funktioniert. Docker sollte man sicherlich kennen. Generell sollte man eine CI / CD Pipeline einrichten können. Man sollte sich mit einer Plattform wie GitHub oder GitLab auskennen, was über die reinen Git Befehle hinaus geht. Diese Dienste bieten dann zum Teil auch an, dass man die Build Pipeline dann dort konfiguriert oder dass man z.B. sowas wie die GitHub oder Gitlab pages benutzt, also saubere Dokumentationsgenerierung. Das wäre so die Tool Chain, die im Groben aus meiner Sicht zentral ist für modernes oder aktuelles Coding. So etwas wie Spring habe ich jetzt rausgelassen, weil ich nicht weiß, ob du das als Werkzeug sehen würdest.

I: [01:50] Ja, ich hatte da schon in die Richtung gedacht, die du auch eingeschlagen hattest.

B: [01:57] Gut, ich finde das auch nochmal relativ zentral, aber vielleicht kommt das noch in einer anderen Frage.

I: [02:08] Du hattest eben modernes Coding erwähnt. Was macht denn aus deiner Sicht überhaupt modernes Coding aus?

B: [02:24] Also, dass man seinen Code so schreibt, dass er nachhaltig ist und von einem selber, aber auch von Kollegen oder sonstigen Mitarbeitenden noch einen Monat nach Erstellung, ein halbes Jahr nach Erstellung oder auch zwei Jahre nach Erstellung verstanden werden kann. Das hat unmittelbar etwas mit der Wartbarkeit des Codes zu tun. Das hat aber auch damit zu tun, dass der Code auf der untersten, atomaren Ebene, letztlich zu

einer guten und auch nachhaltig wartbaren Architektur beiträgt.

I: [03:16] Du gehst also in die Richtung von Qualitätskriterien und machst das daran fest?

B: [03:26] Was mich am meisten überzeugt aus der Literatur sind die Ansätze von Bob Martin aus dem Clean Code, dass man nämlich so ein paar leicht merkbare einsichtige Regeln für guten Code definiert. Ich finde auch so einen Ansatz wie Test-Driven Development sehr sinnvoll und gut, bei dem man im Prinzip die Funktionalität, die man implementiert, erst mal in Form von Tests abbildet, damit man nachher in kleinen Schritten gegen diesen Test implementieren kann. Wenn ich TDD mache, verlagere ich also Iterationen von Fachlichkeit zur Implementierung in ganz kleine Schleifen. Das empfinde ich als ein sehr sinnvolles Vorgehen und ich mache das auch selber, wenn ich programmiere. Und dann könnte man noch folgendes ergänzen, denn du hattest ja gefragt, was modernes Coding ist: Ich würde sagen, auf der etwas höheren Warte gehört für mich auch dazu, nach einem Ansatz wie dem Domain Driven Design vorzugehen. Dass man also die Prinzipien einhält, die das TDD fordert, also primär versucht, die Fachlichkeit angemessen zu durchdringen und zu verstehen. Dass man auch versucht, die Fachlichkeit in abgegrenzte Kontexte zu zerteilen, sodass man dann ohne wahnsinnig viel Absprache autark arbeiten kann als kleines Team. Dazu gehört auch, dass man akzeptiert, dass man in Iterationen arbeiten muss. Also dieses Paradigma “Getting it right the first time“ als Paradigma von gestern wahrnehme und einfach eher davon ausgehe, ich muss mehrere Iterationen machen, ich muss die schnell hintereinander machen. So hat man mehr Zeit, ein Problem mehrfach zu durchdenken, bis man es dann wirklich verstanden hat.

I: [05:57] Ich hätte eine generelle Frage, auf Programmiersprachen bezogen. Es gibt ja viele Programmiersprachen, die auch als modern gelten, deswegen wäre meine Frage, was für Programmiersprachen fallen dir ein, welche viele Paradigmen vereinen, die sehr modern sind?

B: [06:22] Die Diskussion, die ich eigentlich mit Kollegen immer habe, wenn es darum geht, wie mache ich Basis Coding Kurse, ist dann eher die Frage, ob wir es in Java oder in Kotlin machen. Das sind eigentlich so die beiden Pole, die mir da im Moment am ehesten einfallen. Ich nehme wahr, dass eine große Menge von JVM basierten Sprachen existieren, die alle so ihre Meriten haben. Also ich habe vor 15 Jahren mal ein größeres Projekt in Groovy gemacht, war da auch ein großer Fan von, weil Groovy einer der ersten Vertreter von diesen JVM Sprachen war, kompatibel zu Java, aber ohne diesen ganzen Boilerplate Ansatz, den man bei Java so hat. Mich hat das sehr überzeugt und es hat mir sehr viel

Spaß gemacht. Von daher verstehe ich auch die Faszination für Kotlin. Ich finde, dass die Leute, wenn sie von uns weggehen, auf jeden Fall Java können müssen, denn immer noch ein Großteil der Projekte, die gemacht werden, werden in Java gemacht. Und ich finde auch, dass man sich sozusagen eine Haltung angewöhnen sollte, dass man nicht nur einfach Sprache X oder Sprache Y kann, sondern dass sozusagen das Erlernen und Ausprobieren neuer Sprachen, abhängig auch von der Problemstellung, essentiell zum guten Codieren dazugehört. Man sollte also eine Sprache wie Rust auch mal auf dem Schirm haben und schauen, ob die Sprache z.B. gut funktioniert, wenn man etwas performantes braucht. Oder vielleicht gibt es mal eine Anwendung, die ich vielleicht in Django machen möchte, um zu schauen, ob das richtig schön ist. Also ich finde, das ist eine Haltung, die auch zum Codieren dazugehört.

I: [08:26] Du sagtest, Kotlin ist eine moderne schöne Sprache. Java lehrt man eher, weil es sehr weit verbreitet ist. Was würdest du aber jetzt höher priorisieren in der Lehre? Würdest du eher Sprachen lehren, die sehr weit verbreitet sind oder eher Sprachen lehren, die ebenso wie Kotlin sehr modern sind?

B: [08:47] Ich bin schon ein großer Java Fan. Ich wurde früher mit Turbo Pascal sozialisiert, so alt bin ich dann schon. Das ist das erste, was ich im Studium gelernt habe, dann kam Modula zwei. Danach musste ich harte Jahre von Fortran erleiden, also auch die Objektorientierung von Fortran in Richtung Fortran 90. Das war eher halbgar und dann kam schon Java. Ich war nie so ein Microsoft Mensch, diese C# Richtung habe ich nie aktiv gemacht. Das heißt also, ich habe mehr oder weniger mein ganzes aktives Programmiererleben und die Zeit, die ich als Architekt verbracht habe, wo ich dann nicht mehr so viel programmiert habe, nur in Java programmiert. Von daher bin ich da schon ein Fan und finde Java auch nicht unmodern. Es hat eben ein paar Schwächen. Es ist halt sehr groß und manchmal auch sehr umständlich. Diese moderneren Sprachen sind dann eleganter in bestimmter Hinsicht. Und deswegen finde ich es auch schön und sinnvoll, sich damit zu beschäftigen. Also wie gesagt, Groovy hat mir wahnsinnig viel Spaß gemacht, ich fand das ganz toll. Ich finde es auch jetzt immer noch toll, mal was Neues auszuprobieren. Auch wenn man bestimmte Sprachen nicht so richtig gut findet, sollte man sie sich trotzdem, wenn man die Zeit und die Gelegenheit hat, mal anschauen. Ich denke da z.B. an sowas wie Python, weil es im ganzen Machine Learning Umfeld weit verbreitet ist. Der Ansatz “Keeping an open mind“ wäre mir an dieser Stelle sehr wichtig, auch für mich selber. Mit Blick auf Leere würde ich aber schon priorisieren, dass die Leute mit der aktuellen Brot und Butter Sprache definitiv auch am meisten konfrontiert werden. Das ist im Moment nach wie vor Java. Oder z.B. auch C#, wenn man sagt, wir bilden die

Leute für einen Microsoft Weg aus. Und daneben vermittelt man die Haltung, dass wenn man eine Sprache kann, dann kann man auch ganz viele oder man kann sie sich schnell beibringen.

I: [11:12] Als ich eben gefragt hatte, was für dich modernes Coding ist, hattest du Clean Code erwähnt. Und zwar meine Frage wäre jetzt, wie kann man denn Lernende dazu bringen, dass sie sich auch an Richtlinien halten, welche zu sauberen Code führen?

B: [11:32] Indem man es zum Thema macht. Ich bin eigentlich nach wie vor sehr glücklich damit, wie wir das in Code & Context gemacht haben. Einige Kollegen und ich führen diesen Studiengang sehr Werkstatt orientiert und sehr Hands on. Bei Clean Code habe ich die gerade berufene Professur für Agile Coding vertreten und wir haben letztlich das Programmierbeispiel wieder aufgenommen, was wir bei Coding Essentials zwei gemacht haben, wo du auch mit dabei warst. Da haben die Studierenden ja zum Schluss ein Text Adventure geschrieben, und da kamen irgendwie so 10 Texte Adventures dabei raus. Und in Clean Code sind wir dann natürlich die Inhalte durchgegangen, kleine Übungen gemacht, und haben ihnen dann die Aufgabe gegeben: Jetzt nehmt euch nochmal euren Code von vor einem halben Jahr vor und schaut ihn euch nochmal kritisch an und überarbeitet den mit dem, was ihr hier gelernt habt. Beachtet also mal die Regeln aus dem Clean Code und versucht dabei mal TDD anzuwenden. Das war total effektiv. Von den Studierenden kam ganz viel Feedback, dass diese jetzt sehen, dass sie ihren eigenen Code nicht mehr verstanden haben und jetzt sehen, was jetzt besser ist als vor einem halben Jahr. Solche Veranstaltungen würde ich mir gerne mehr wünschen. Das hat im Moment in dem Informatik Bachelorstudiengang bei uns nicht richtig eine Heimat. Das finde ich sehr schade.

I: [13:35] Du meinst quasi, man macht es in extra Fächern zum Thema und zeigt den Leuten, was es bringt und was man daraus lernen kann. Kennst du praktikable Wege, wie man das in Aufgaben, die die Studenten lösen, auch erzwingen kann, wenn man jetzt nicht diese Kapazitäten hat, das in extra Fächer zu verpacken?

B: [13:59] Man kann es natürlich versuchen als Inhalte in die existierenden Fächer reinzubauen. Wir hätten ja Fächer, wo man das durchaus zum Thema machen kann oder vielleicht wird es auch teilweise schon zum Thema gemacht. Ich kann da jetzt für die Kollegen nicht sprechen. Also vielleicht ist es bei Algorithmen und Programmierung oder bei Paradigmen der Programmierung zum Teil mit drin. Jetzt hast du gefragt, wie man das erzwingen kann. Man kann es wie gesagt ein bisschen zum Thema machen und dann

kann man natürlich versuchen, den Weg zu gehen, den wir hier ja auch in Softwaretechnik beschritten haben. Dass man eben versucht, den Code automatisch zu prüfen. Also erst mal überhaupt die Studierenden im Praktikum Code schreiben lässt und den Code nicht mehr manuell nachguckt, sondern automatisch nachprüfen lässt. Einfach damit es effizient ist und damit man sich nicht in Arbeit verliert. Und dann kann man natürlich auch hingehen und mit SonarQube bestimmte Qualitätsmetriken aufstellen. Manches wird da leichter sein, manches wird nicht so leicht sein. Z.B. eine Methodenlänge oder eine Klassenlänge zu bewerten wird gehen, aber die Vergabe von sinnvollen Methodennamen und Variablennamen wird wahrscheinlich schwierig automatisch zu prüfen sein.

I: [15:57] Jetzt hast du ja schon mal ein paar Richtlinien genannt? Was wären denn da konkrete Richtlinien, die du besonders wichtig finden würdest, dass man diese überprüft, wenn es denn möglich ist?

B: [16:26] Also die Methodenlänge wäre schon mal definitiv ein Punkt. Das wäre wahrscheinlich relativ leicht rauszukriegen, wenn man nochmal über die einzelnen Bestandteile des Clean Code Ansatzes drüber geht und schaut, was man denn mit vertretbarem Aufwand umsetzen kann. Und vielleicht auch schauen, was schon in dem Standard Regelsatz von SonarQube enthalten ist. Das wäre jetzt eher eine Fleißarbeit, finde ich. Das fällt mir aus dem Stehgreif ein bisschen schwer.

I: [17:19] Du hattest eben auch nochmal Domain Driven Design erwähnt und wie wichtig das ist. Aber Domain Driven Design ist ja auch eine Herangehensweise für eigentlich komplexere Software, die man auch gut in einzelne Bereiche zerteilen kann. Wenn man das jetzt aber lehren möchte, hat man oft das Problem, dass die Übungsaufgaben von der Komplexität ja eher geringer ausfallen, weil man auch einfach diese Kapazität nicht hat. Wie kann man denn dann durch diese Übungsaufgaben trotzdem einen guten Lerneffekt erzielen?

B: [18:07] Ich denke, dass man beim Tactical Design eher noch ein leichteres Spiel hat als beim Strategic Design. Die Building Blocks lehren wir ja auch jetzt schon. Man kann die Unterscheidung zwischen Entity und Value Object abfragen. Man kann die Studierenden trainieren, in sowas wie einer Ubiquitous Language zu denken und mit Begriffen sehr genau umzugehen. Auch das machen wir jetzt schon, indem wir quasi ein fachliches Glossar erstellen lassen. Mit den jetzigen Mitteln könnten wir auch automatisch prüfen lassen, ob das Glossar die richtigen Begriffe enthält. Ich würde die Aufgabenbeschreibung etwas länger machen als sie es jetzt bei uns war. Jetzt war sie ungefähr eine halbe Seite lang und

da habe ich an den Rückfragen der Studierenden sehr oft gemerkt, dass da zu viel Gewicht auf einzelnen kurzen Formulierungen war. Dies ist zu anfällig für Missverständnisse. Wenn man die Studierenden ein bisschen mit der Nase drauf stoßen will, was ist denn z.B. jetzt an Value Object oder ein Entity, dann sollte man das schon ein bisschen ausführlicher beschreiben, sonst ist es einfach wirklich zu schwierig. Und wenn man zum Beispiel die Studierenden dazu bringen will, Aggregatgrenzen festzulegen, dann muss man auch da eigentlich mehr Text haben. Wenn ich in Richtung Strategic Design gehe, also z.B. einen Bounded Context Schnitt mache, dann habe ich das Gefühl, dass das im Rahmen von Vorlesungen wirklich sehr schwierig ist. So etwas wird man auch nicht mehr gut automatisch abprüfen können. Da wäre ich eher der Ansicht, dass man den Schnitt eher vorgibt. Meine Lehre funktioniert im Moment so, dass ich diesen Teil eher im Master mache und auch da häufig diesen ersten Schritt des Schnitts aus Zeitgründen nicht mache. Dabei fällt generell dieses iterative Prinzip des DDD mehr oder weniger komplett über Bord, weil man dafür nicht die Zeit hat. Also eigentlich müsste ich ja sagen, ich werfe etwas einmal weg, zweimal weg, dreimal weg, mache es dreimal neu und habe eine Fachseite, die so geduldig ist, dass sie immer wieder alle Fragen beantwortet. Ich wüsste nicht, wie das in der Lehre zu realisieren wäre.

I: [21:12] Aber das ist ja schon mal ein Ansatz, dass man einfach gewisse Vorgaben macht. Man ist zwar ein bisschen eingeschränkt, aber hat trotzdem die Möglichkeit, überhaupt Übungen zum Tactical Design zu machen. Ich würde jetzt mal in Richtung Kooperation und Organisation von Praktika übergehen. Wenn man jetzt davon ausgeht, man würde Übungsaufgaben auch bewerten, würdest du da eher Einzelarbeit oder Gruppenarbeit bevorzugen?

B: [21:47] Da habe ich eine ganz klare Haltung mittlerweile zu entwickelt. Ich möchte Gruppenarbeit, aber einzelne Abgaben haben. Und dann würde ich sehr gerne an dem Thema der Individualisierung noch weiterarbeiten. Wir haben jetzt ein Maß an Individualisierung zwischen den verschiedenen Varianten, das doch noch relativ leicht durchschaubar ist. Das ist schon mal nicht schlecht, aber das hätte ich gerne noch ein bisschen besser. Also der eine Gedanke ist: Ich möchte keine Gruppenarbeiten mehr. Ich bin so verbrannt von Abgaben in diesen herkömmlichen Praktika. Da hast du immer einen Wortführer, der immer alles für die ganze Gruppe erzählt und dann versuchst du irgendwie ganz mühsam Sonderregeln einzuführen, dass z.B. jetzt auch mal jemand anders vorstellen muss. Man merkt dann ganz deutlich, dass derjenige sich das eine Viertelstunde vor dem Praktikumstermin mal eben angeschaut hat und kann so gerade die wesentlichen Dinge erklären. Eigentlich müsstest du ihm das rechts und links um die Ohren hauen, aber das

machst du eben nicht, weil es dann auch nicht wirklich quantifizierbar ist. Du hast zwar das Gefühl, wirklich tief bohren kann ich hier nicht, aber er hat eigentlich formal die Anforderungen erfüllt, die du gestellt hast. Ich habe da einfach keine Lust mehr drauf und das ist nicht effektiv. Ich verstehe, dass die Leute dazu neigen, arbeitsteilig vorzugehen. Das Studium sorgt zwar für eine große Belastung, aber trotzdem ist das Ergebnis der Studenten oft überhaupt nicht zufriedenstellend. Deswegen finde ich das Prinzip, so wie wir es jetzt fahren, deutlich besser. Also die müssen alle ihre individuelle Lösung abgeben, aber natürlich arbeiten die in Gruppen zusammen und wir ermutigen sie auch, das zu tun. Die sollen diskutieren und auch gerne ihre Lösungen vergleichen und dann sehen sie eben die Ähnlichkeiten. Aber sie müssen es immer noch selber umsetzen, weil sie sonst keine grünen Tests bekommen. Und ich glaube, das ist so der der Weg, der sinnvoll ist. Und gerne hätte ich noch eine Individualisierung, die dieses Storytelling bei der Aufgabenstellung noch ein bisschen vereinfacht. Wenn man z.B. Beziehungstypen individualisiert, ist es dennoch sehr schwierig, dies vernünftig in den Aufgabentext zu transportieren. Man möchte ja eine Story erstellen, die konsistent bleibt, egal ob eine Beziehung jetzt 1 zu n oder n zu m ist. Wenn wir so weit sind, dann wäre ich wirklich zufrieden.

I: [24:45] Zusammenfassend findest du auf jeden Fall Gruppenarbeit super, aber hast trotzdem lieber einzelne Abgaben. Was würdest du denn da von Pair-Programming in Abgrenzung zu Gruppenarbeit halten?

B: [24:59] Ich finde, das ist auch ein sehr spannender Ansatz. Wir haben bei Code & Context ein paar Mal versucht, die Leute zu Pair-Programming zu animieren. Die Leute haben sich dann in Peers zusammengefunden und wir haben ihnen dann die Regeln für Pair-Programming erklärt. Das hat bei Code & Context relativ gut funktioniert. Bei Code & Context haben wir eine Gruppe von ca. 20 Personen im ersten Jahrgang, die als Gruppe gut funktioniert haben. Da war also so ein gewisses Maß an Gruppendruck da und die haben sich gegenseitig auch ein bisschen dazu animiert, solche Vorgaben auch einzuhalten. Das ist in der allgemeinen Informatik bei uns alles ein bisschen anonymer und da müsste man noch mal einen Weg finden, wie man sowas wie Pair-Programming tatsächlich erzwingen kann in der Bearbeitung. Da sehe ich im Moment nicht wirklich einen Weg, wie ich das aktiv gestalten kann. Wenn ich ein Setting definieren könnte, wo 90 Prozent oder 80 Prozent der Leute dabei mitmachen, wäre ich auch schon zufrieden. Dann müsste ich es gar nicht erzwingen. Aber auch da wüsste ich nicht, wie ich das machen sollte. Wenn ich einfach nur sage, wir machen 50 Sprachkanäle auf und geht mal bitte auf die 50 Sprachkanäle, dann weiß ich nicht, wie viele von denen das wirklich mitmachen würden. Da müsste man glaube ich noch ein bisschen mehr aktive Hinführung betreiben.

I: [26:47] Man hört ja heraus, dass man Studenten oft Druck machen muss, damit diese eben auch mal was machen. Es gibt ja verschiedene Arten, diesen Druck zu erzeugen. Eine Möglichkeit wäre ja, den Druck durch Individualisierung, Noten oder durch bestanden, nicht bestanden zu erzeugen. Aber auf der anderen Seite kann man auch manchmal ein motivierendes Lernklima erzeugen, bei dem man diesen künstlichen Druck vielleicht nicht braucht. Die Frage ist jetzt, unter welchen Umständen meinst du, kann welcher Ansatz funktionieren und sollte dann auch verfolgt werden?

B: [27:34] Das ist wirklich die absolute Schlüsselfrage. Also bei Code & Context merken wir, dass es mit dem ersten Jahrgang ganz ordentlich funktioniert hat. Wir sind jetzt im zweiten Jahrgang, da sind es jetzt ca. vierzig Personen. Ich habe einen Kurs mit denen zusammen gemacht, das war Computational Thinking und das hat auch ganz gut funktioniert. Das ist jetzt so ein Datenpunkt und reicht natürlich noch nicht wirklich für eine Bewertung. Also da spielt so dieser Werkstatt Charakter eine Rolle und der Fakt, dass die als ein geschlossener Jahrgang immer durch diese einzelnen Blöcke durchgehen. Die verbringen eigentlich viel mehr Zeit immer in der gleichen Konstellation als das unsere Studierenden tun. Man könnte auch sagen, das Maß an Verbindlichkeit ist viel höher. Nicht durch zwangsweise Vorgaben, sondern einfach dadurch, dass wir eine Tagesstruktur definieren und die Studierenden dann einfach mitgehen, weil sie sozusagen diese Tagesstruktur einfach akzeptieren und annehmen. Also wenn wir es hier in der allgemeinen Informatik auch schaffen würden, mehr im Block zu arbeiten, wäre ich sehr zuversichtlich, dass wir es auch schaffen würden, so ein Lernklima stärker herzustellen. Ich bin durch Code & Context, aber vorher auch schon, wirklich bestärkt worden, dass Block Lehre richtig gut ist. Es tut einfach gut, mal weg von Task-Switching zu gehen und stattdessen mal für längere Zeit in einem Thema zu bleiben.

I: [29:28] Du hattest ja eben das Thema Individualisierung angesprochen. Was sind dir denn so für Individualisierungsmethoden bekannt, mit denen man Aufgaben in einem Maß individualisieren kann, sodass ein einfaches Kopieren gar nicht mehr möglich ist?

B: [29:41] Also Ilias liefert ja zum Beispiel diese Fragenpools, wo du dann irgendwie eine Menge an Aufgaben definierst und diese dann zufällig den einzelnen Personen zugeordnet werden. Das würde definitiv gehen. Darüber hinaus habe ich tatsächlich nicht groß recherchiert, was es sonst noch an fertigen Lösungen gegeben hätte. Als wir darüber nachgedacht haben, wie wir das machen können für Softwaretechnik, da ergab sich aus meiner Sicht relativ schnell, dass wir eine ziemlich spezielle Aufgabenstellung vor uns hatten. Ich hätte

nicht gewusst, wo man da was hätte suchen können. Ich denke, den Ansatz, den wir da gewählt haben, der ist schon ganz gut angepasst auf das, was wir in dieser Veranstaltung machen wollen. Diese Mischung umfasst fachliche Analyse, die ein bisschen vom Geist des DDD geprägt ist, Modellierung und andererseits die Umsetzung dessen in Code. Dafür brauchst du glaube ich schon eine maßgeschneiderte Lösung und ich finde, da sind wir auf einem ganz guten Weg mit der Implementierung dieser Lösung.

I: [31:15] Was wäre aus deiner Sicht wichtiger: Dass man schaut, dass man wirklich einen hohen Grad an Individualisierung schafft oder dass man trotzdem noch eine sehr konsistente Geschichte hat. Du sagtest ja eben, du würdest auch lieber längere Aufgabentexte haben. Ich würde das aber trotzdem gerne mal fragen: Welcher Ansatz von den beiden wäre also für dich wichtiger?

B: [31:38] Ich glaube, ich würde die Konsistenz der Geschichte höher bewerten, weil es absolut mein Anspruch ist, dass die Leute in meinen Modulen ST1 und ST2 vom gesprochenen Wort anfangen. Sie sollen bei den Anforderungen anfangen und nicht sofort bei der Technik ansetzen. Es zieht sich ja wirklich eigentlich durch alles durch, was ich mit Studierenden mache, dass ich bei der Fachlichkeit ansetze. Und wenn ich dann nur so eine Sammlung von technisch geprägten Aussagen habe, dann ergibt sich kein fachliches Bild. Die allererste Version bei ST2 war ja so geprägt, dass wir das sehr technisch formuliert haben. Und damit bekommt man diesen ersten Schritt von der Fachlichkeit zur Technik nicht hin. Schön wäre es natürlich, um es nochmal zu sagen, wenn man beides hinkriegen könnte.

I: [33:05] Auch wenn man beides will, hat man leider immer einen kleinen Zielkonflikt zwischen den zwei Sachen. Also selbst wenn man den Zielkonflikt weitestgehend auflösen würde, hätte man wahrscheinlich doch mehr Möglichkeiten, wenn man die Fachlichkeit außen vor lässt.

B: [33:19] Also dann würde ich da gerne noch etwas anschließen. Also für die Weiterentwicklung des Toolings auf längere Sicht würde ich mir wünschen, dass wir da das Schreiben von Ausgangstexten etwas erleichtern. Durch Platzhalter zur Individualisierung in Form von Variablen werden zum Beispiel die Texte fast unlesbar im Rohformat. Und ich glaube, dass man da durchaus nochmal nach Ansätzen schauen könnte. Da würde ich auch nochmal ein bisschen in die Recherche gehen, was es da so sonst noch so gibt an irgendwelchen Markup Sprachen, die einfach dieses Schreiben von Geschichten leichter machen. Also ich weiß, dass mir persönlich das ganz gut liegt. Ich kann ganz gut so Geschichten schreiben,

die dann in Aufgaben münden. Ich denke mir auch sehr gerne Geschichten aus, die man dann in Code umwandeln könnte. Und da würde ich mir gerne mehr Potential wünschen, die Kreativität austoben zu lassen. Weil dann kannst du komplexere Geschichten erzählen. Und wenn sie dann noch ein bisschen stärker individualisierter sind, ist das gut, aber wenn nicht, sind diese zumindest erst einmal so komplex, dass sie der Wirklichkeit näher kommen.

I: [34:45] Jetzt würde ich mal zu einem Problem von Individualisierung übergehen. Es kann ja sein, dass durch Individualisierung der Schwierigkeitsgrad abweicht zwischen einzelnen Aufgaben, die verschiedene Lernende bekommen. Wie ist deine Meinung dazu?

B: [34:58] Das ist natürlich blöd. Bei Softwaretechnik 1 und 2 hatten wir mal ein Restaurant-Managementsystem als Beispiel. Da haben wir dann verschiedene Bounded Contexte definiert und diese haben wir dann den einzelnen Praktikumsgruppen zugeteilt. Und das war dann doch sehr unterschiedlich schwer. Die eine Gruppe hatte im Statusdiagramm dann z.B. zwei Status und die andere Gruppe hatte acht Status, das waren so die Extreme. Dafür war das dann aber bei anderen Aspekten der Aufgabenstellung wieder anders gemischt. Richtig zufrieden hat mich das nicht gemacht. Also da hätte ich gerne eine etwas höhere Konsistenz und da habe ich das Gefühl, dass der Ansatz, den wir jetzt fahren, besser ist. Also dass jeder quasi eine wirklich auf ihn oder sie selbst zugeschnittene Aufgabe bekommt. Weil dann hat man wirklich eine relativ homogene Verteilung des Schwierigkeitsgrade, natürlich mit dem Preis, dass die einzelnen Lösungen eben nicht so stark voneinander abweichen. Und wenn dann jemand mit kritischem Blick drauf guckt, wird er verstehen, wie die Individualisierungen zustande kommen und Lösungen kopieren können. Auf der anderen Seite würde ich damit dann im Zweifel leben, weil dann heißt es immer noch, dass ich mir die Lösung von jemand anders anschauen, diese verstehen und dann auf meine Lösung übertragen muss. Damit hätte ich schon einen Grad an Mindestverständnis garantiert, was mir als Praktikums Ergebnis schon genügen würde.

I: [37:10] Du hast eben erwähnt, dass das Schreiben von Aufgabentexten sehr schwierig sein kann, wenn es um eine Aufgabe geht, die auch individualisiert werden kann. Wie viel Mehraufwand wärst du bereit zu investieren, eben solche Texte zu schreiben im Vergleich zu Texten von einfachen ganz normalen Aufgaben?

B: [37:33] Also ich habe das bis jetzt schon zwei Mal gemacht. Einmal für die ST2 Klausur und einmal für ST1, das war jedes Mal ein Gesamtkunstwerk. Das schwierige ist, dass du eine Geschichte von hinten erdenken musst. Also insofern, dass man überlegt, welche

Aspekte will ich am Ende noch alle drin haben. Ich fange letztlich bei der Musterlösung an. Erstelle also eine Art generische Musterlösung und entwickle die dann von hinten nach vorne, bis ich dann dazu einen Aufgabentext schreiben kann. So ein Entwicklungsprozess ist aufwendig, aber das ist aus meiner Sicht gut investierte Zeit. Weil mir würde kein besserer Weg einfallen, um am Ende den Studierenden eine Aufgabenstellung zu präsentieren, die relativ nah an dem ist, was man dann später bei einem richtigen Kunden auch zu Gesicht bekommen würde in Form eines Anforderungsdokuments.

I: [38:54] Ich würde jetzt mal in Richtung Automatisierung übergehen. Zuerst einmal eine ganz generelle Frage: Was müssen Übungsaufgaben für Eigenschaften aufweisen, damit man diese überhaupt automatisch überprüfen kann?

B: [39:23] Man braucht eine gewisse Eindeutigkeit der Antwort. Dazu zählen Fragen wie z.B. “Was ist ein Entity, was ist ein Value Objekt?“ oder “Welche Klassen sind im logischen Datenmodell noch enthalten?“. Dann müsste ich sicherstellen, dass ich sinnvolle Fehlermeldungen gebe. Da ist das Tooling, was wir jetzt haben z.B. an einer Stelle noch nicht gut genug. Wenn quasi nur die Meldung zurückkommt, die Tabelle stimmt so nicht, irgendwo ist ein Fehler, dann ist dieses Feedback so nicht ausreichend. Das ist einfach ein Punkt, wo wir weiter dran arbeiten müssten, sodass man an dieser Stelle einen Weg findet, auf das Problem hinzuweisen, ohne jetzt zu viel zu verraten. Die Leute sollen ja auch nicht mit “Trial and Error“ nach zehn Minuten die Lösung gefunden haben. Das ist dann eine Herausforderung, der man sich stellen muss. Was z.B. nicht so gut geht, ist die Bewertung eines Modells, welches die Leute als UML Diagramm hochladen. Da wüsste ich tatsächlich im Moment noch nicht, wie man mit den jetzigen Mitteln sinnvoll automatisch prüfen könnte. In ST2 konnten wir tatsächlich komplett automatisch testen. In ST1, wo es sehr stark noch um diese Anforderungsermittlung und das Aufstellen von Domainmodellen geht, hätten wir wahrscheinlich den manuellen Aufwand auf sagen wir mal 20 bis 30 Prozent drücken können. Im Moment war der manuelle Aufwand leider viel höher, aber wir sind ja in der Entwicklung.

I: [41:32] Ein anderes Problem von automatischer Überprüfbarkeit ist ja, dass man eine gewisse Eindeutigkeit braucht und deshalb auch gewisse Dinge verlangt, welche die Freiheit von Lernenden einschränken, die Übungsaufgaben lösen sollen. Hast du da Ideen, wie man das ein bisschen vereinen kann?

B: [42:27] Man muss schon schauen, dass man eher in Richtung von Contract basiertem Testen geht. Also wenn man mal Rest API’s als Beispiel nimmt, da geht das finde ich

richtig gut. Ich erwarte dann von dem API, dass es mir auf diese Anfrage jene Antwort zurück liefert. Dann könnte ich den Leuten in der Umsetzung erstmal komplett überlassen, wie sie es machen. Dann könnte ich mit gewissen Sonarqube basierten Mitteln vielleicht nochmal überprüfen, ob sie z.B. einen Application Service implementieren. Wenn wir also sagen, also nach DDD gibt es eben diesen Application Layer und implementiere bitte immer einen Application Service und gib z.B. kein Entity nach draußen und so weiter. Das kann man wiederum ganz gut überprüfen. Und wie sie es im Detail machen, könnte man ihnen überlassen.

I: [43:40] Das heißt also, dass man gewisse Vorgaben macht, aber den Rest frei lässt.

B: [43:47] Genau, ich meine Contract basiertes Testing oder auch Black-Box Testing, was so das Testen von Schnittstellen angeht. Dazu würde man noch das Einhalten von Design Prinzipien und Clean Code abtesten. Dann könnte man diese SOLID Prinzipien von Bob Martin dann auch noch mit reinnehmen. Man würde dann diese Black-Box Tests, also diese Contract basierten Tests, um Tests anreichern, welche Architekturprinzipien abdecken. Wenn wir sagen, es ist uns wichtig in der Veranstaltung, dass die Studierenden an dieser Stelle dieses und jenes einhalten, dann gibt es dann da einen Test für.

I: [44:41] Jetzt geht es noch ein bisschen Richtung Tooling. Das deckt dann auch automatisches Testing und Individualisierung ab. Hast du schon mal Tools zur Aufgabenindividualisierung in der praxisorientierten Lehre angewendet? Du hast ja schon ein paar genannt, aber fallen dir noch ein paar ein, die du verwendet hast?

B: [45:14] Ich habe das tatsächlich noch nie vorher gemacht. Dass ich bei Software Technik die Leute auch Coden lasse, ist relativ neu. Ich mache sonst keine Coding Ausbildung, das machen andere Kollegen. Mit diesen Ilias Ansätzen habe ich mal kurz experimentiert, fand diese aber ein bisschen einengend und unkreativ. Ich fand dann eine maßgeschneiderte Lösung irgendwie attraktiver. Aber wie gesagt, richtig systematisch recherchiert habe ich das tatsächlich nicht, weil mir der Glaube gefehlt hat, dass sich so eine Recherche lohnt und dass man da was von der Stange findet, was dann so gut auf die eigene Lehrveranstaltung passt.

I: [46:05] Wir haben ja eben auch viel über automatische Überprüfung gesprochen. Inwieweit meinst du, kann Feedback von automatischer Überprüfung das Feedback von manueller Korrektur oder Lehrenden ersetzen?

B: [46:21] Ich glaube, dass es das nicht ersetzen kann. Also man sollte unbedingt solche automatisch abgenommenen Praktikumsleistungen flankieren mit Beratungsstunden. Auch da bin ich eigentlich wieder ganz zufrieden mit diesen Discord Sessions, die wir gemacht haben. Wo wir dann gesagt haben, wir sind einfach als Betreuungsteam da und machen vielleicht nochmal so kurze Impulse von Sachen, die man öfter mal sieht und die falsch gemacht wurden. Oder man macht gezielt Übungen zu neuen Inhalten. Und dann gibt es aber auch eine Phase, wo sich die Leute einfach in diesen Sprachgruppen treffen können und das Betreuungsteam herbeirufen, sodass man Probleme mit denen bespricht und inhaltliche Fragen klärt. Das funktioniert aus meiner Sicht online wirklich mehr oder weniger genauso gut wird es live funktionieren würde. Also live würde man ebenso einen großen Raum nehmen, wo sich die Leute an verschiedene Tische setzen können. Oder sie sitzen in verschiedenen kleinen Räumen in Kleingruppen zusammen. Ob man nun virtuell sozusagen von Team zu Team geht oder in einem physischen Raum von Team zu Team geht, ist fast egal. Das funktioniert glaube ich beides ganz gut.

I: [47:41] Wir haben jetzt viel über Übungsaufgaben gesprochen und diese müssen auch erstmal irgendwie an Studenten verteilt werden und am Ende muss man ja die Lösung auch wieder einsammeln. Und wenn man dann eine ganze Menge an Studenten hat, kann das echt viel Aufwand sein. Das heißt, dass man ja oft eine Plattform hat, über die das abläuft. Mit was für Plattformen hast du da Erfahrungen gemacht, über die man das realisieren kann?

B: [48:05] Also eigentlich mit Ilias und dann das, was wir selber machen, das also auf Gitlab basieren. Damit bin ich total zufrieden. Das würde ich gar nicht anders haben wollen. Also erst einmal funktioniert es einfach gut. Man bekommt dahingehend viel von der Stange, hat aber andererseits einen großen Teil der Toolchain auch unter eigener Kontrolle. Also wenn man irgendwas ändern will, dann geht das sehr gut. Und zum zweiten trainiert man den Leuten schon frühzeitig an, wie sie professionelle Software entwickeln. Das ist ein Nebeneffekt, den würde ich gar nicht unterschätzen. Siehe der Anfang des Interviews, wo wir über notwendiges Tooling für modernes Coding gesprochen haben. Dass man nicht mehr Dateien auf dem C-Drive ablegt oder irgendwie durch die Gegend schickt, sondern eben mit einem VCS arbeitet. So etwas gehört auch dazu, dass man eben sein Handwerkszeug beherrscht, wenn man rausgeht aus der Hochschule.

I: [49:10] Du hast ja eben auch mal Discord angesprochen. Das lag jetzt vor allem auch an Corona, dass wir auf Discord umgestiegen sind. Was hältst du generell von modernen Messengern als Kommunikationskanal auch außerhalb von Corona Bedingungen?

B: [49:27] Ich bin total angetan davon, wie leichtgängig das funktioniert. Also erstmal passt es super zu der Altersgruppe, da würde ich quasi die Betreuer noch mit reinnehmen. Also die Leute, die so ganz grob zwischen 16 und 29 sind, sind glaube ich total mit Discord sozialisiert. Und andererseits bekomme ich es ja bei den Kollegen aus dem ArchiLab auch noch mit, da geht die Altersspanne ja doch bis über 30 und auch da scheint Discord völlig selbstverständlich zu sein. Also "jeder" ist mehr oder weniger "immer" da unterwegs. Ein Plus von Discord ist, dass vieles ganz einfach geht. Also ich kann ganz einfach einen Sprachkanal definieren. Ich kann ganz einfach einem Video Call beitreten und bin auch ganz schnell wieder raus. Das macht es unheimlich leichtgewichtig, was ganz toll ist für solche Beratungsleistungen. Also ich könnte mir auch für die Zukunft vorstellen, wenn diese Corona-Pandemie Geschichte ist, dass man einfach Lehrveranstaltungen hybrid aufsetzt. Also sagen wir mal, man macht Kickoff-Veranstaltungen durchaus persönlich, damit man sich auch mal gesehen hat und die Gesichter zuordnen kann und dann aber durchaus solche Sessions auch über Discord macht. Weil das einfach gut funktioniert und ich das Gefühl habe, dass es zumindest meiner Arbeitsweise sehr entgegen kommt, weil ich keine Regelstunden habe. Also es kann sein, dass ich mal nachmittags was anderes mache, dafür dann aber abends vor dem Rechner sitze. Ich habe auch bei den wissenschaftlichen Mitarbeitern das gleiche Gefühl, dass sie nämlich ähnlich ticken und auch in ihrer Mehrzahl nicht unbedingt an einem "Nine to Five" Job interessiert sind, sondern das gerne dann machen, wenn es für sie passt. Und auch dafür finde ich das eine super Sache, dass man sagt, wenn die Mitarbeiter das auch okay finden, dann machen wir eben weniger feste Beratungstermine und beantworten sonst so, wie es gerade passt. Das Feedback der Studierenden lässt ganz deutlich darauf schließen, dass sie das gut finden.

2.2.4 Experteninterview 4

Interviewpartner: Marco Reitano
Datum: 21.01.2021
Ort: Zoom-Meeting

00:00–55:37

I: [00:02] Also wir fangen jetzt erst einmal ein bisschen auf einer höheren Ebene an. Was sind deiner Meinung nach die wichtigsten Werkzeuge, die man braucht, wenn man jetzt modernes Coding betreiben will?

B: [00:30] Auf jeden Fall Git, da gibt es mittlerweile keinen Weg drum herum. Irgendeine Form von IDE und damit schließe ich eigentlich so ziemlich jede mit ein, die es gibt. Es könnte z.B. Vim, VS Code, IntelliJ oder Eclipse sein, aber eben eine, in der man sich auskennt und in der man fluide ist. Für Programmieranfänger eignen sich meiner Meinung nach eher leichtgewichtigere IDEs wie VS Code, da IDEs mit großem Funktionsumfang oft eher verwirren und ablenken. Und wenn es nicht nur um Eigenentwicklungen zuhause alleine geht: In irgendeiner Form Software Tools, die Teamarbeit unterstützen, also sowas wie ein Kanban Board. Dann auch Absprache Tools im Sinne von einem Miró Board oder einem Trello Board. Das hängt dann immer stark davon ab, wie das Team aufgebaut ist und wie die Teamarbeit aussieht. Wenn das nicht vor Ort ist und kein Whiteboard zur Verfügung steht, braucht man eben solche Tools. Es kommt immer auf das Feld an, in dem man sich bewegt, beispielsweise Docker als grundlegendes Tool in der Backendentwicklung.

I: [02:02] Ich hätte da nochmal eine speziellere Frage, du hattest ja auch Git erwähnt. Würdest du es eher bevorzugen, wenn die Leute Git auch von Anfang an über die Funktionen der IDE nutzen oder sollten sie das lieber über die Konsole machen?

B: [02:20] Ich habe es bis jetzt immer so gemacht, das auf jeden Fall in der Konsole beizubringen. Was sie später machen ist mir egal. Aber ich habe immer die Erfahrung gemacht, dass es besser ist, das den Leuten in der Konsole beizubringen, weil die Konzepte, die dahinterstecken, klarer abgetrennt sind. Also es gibt in der IDE Tools, die teilweise allein schon falsche Begrifflichkeiten nutzen. Also da hast du nicht gepusht, sondern Git Sync ausgeführt. Und die IDEs verstecken teilweise Dinge. Also da muss man dann eben nicht mehr Adden, Commiten und Pushen, sondern das wird mit einem Tastendruck für einen erledigt. Und so fällt es den Studenten dann schwerer, die einzelnen Konzepte dahinter zu verstehen. Und die Konzepte muss man verstehen, um nachher kompliziertere Aufga-

benstellungen erledigen zu können. Ich denke da beispielsweise an Mergekonflikte oder an das Zurückrollen auf einen alten Stand.

I: [03:23] Jetzt nochmal eine generelle Frage, und zwar: Was macht für dich modernes Coding aus?

B: [03:43] Agilität, wenn man das unter Coding verstehen würde. Klein anfangen, sich selbst überprüfen, iterativ arbeiten, kurze Feedback Loops haben. Ich antworte vielleicht auch mit dem Gegenteil: Nicht modernes Coding heißt, sehr lange Dinge tun, die im Verborgenen liegen und dann am Ende in den meisten Fällen mit einem falschen Ergebnis kommen. Und modernes Coding und moderne Entwicklung muss sein, dass man möglichst schnell zu etwas kommt, was überprüfbar ist. Gleichzeitig muss auf eine hohe Qualität geachtet werden und technische Schuld vermieden werden. Themen wie Refactoring und TDD werden dann relevant. Modern bedeutet auch, dass moderne Softwareentwicklung immer mehr ein People Problem und nicht ein technisches Problem ist. Also moderne Softwareentwicklung hat immer mehr mit Kommunikation und Teamarbeit zu tun und nicht nur mit der Frage, wie etwas technisch umzusetzen ist.

I: [05:47] Was fallen dir jetzt so spontan für Programmiersprachen ein, die viele Paradigmen vereinen, welche als modern gelten?

B: [06:50] Ich würde z.B. nicht JavaScript sagen. Wenn überhaupt, dann TypeScript, weil JavaScript ist zwar modern, verachtet aber alles was davor war sozusagen. Also im Endeffekt unterstützen alle Programmiersprachen alles. Es ist eben immer nur eine Sache von Syntactic Sugar und hängt mit der Frage zusammen, wie viel Code ich brauche, um etwas umzusetzen.

I: [07:56] Was würdest du in der Lehre höher priorisieren: Ein Fokus auf weit verbreitete Programmiersprachen oder ein Fokus auf Programmiersprachen, welche auf modernen Paradigmen aufbauen und das Potential aufweisen, auch einmal eine breite Anwendung zu finden?

B: [08:11] Es muss relativ weit verbreitet sein, denn es sollte wie z.B. bei Java viel Material vorhanden sein, welches über das bereitgestellte Material von Dozenten hinausgeht. Wenn man Java in Google eingibt, findet man eine Menge relativ hochqualitative Tutorials, Reference Guides und so weiter. Was JavaScript angeht, ist das mehr als ein großer Schwachpunkt, weil bei JavaScript findet man auch viel, was qualitativ extrem durch-

mischt ist. Das reicht von absoluten Bad-Smell Anleitungen bis hin zu hoch qualitativem Material. Deswegen weit verbreitet und relativ etabliert finde ich gut, weil es sich so ein bisschen festgelegt hat, was denn jetzt gute Programmierung in dieser Programmiersprache ist und was nicht. Gleichzeitig aber auch modern, weil bestimmte Aspekte vielleicht in modernen Programmiersprachen klarer umgesetzt sind. Also wenn man z.B. funktional Programmieren beibringen möchte, würde ich nicht Java nehmen. Und das dritte, was eigentlich dazukommt, ist, dass die Programmiersprache bestimmte Prinzipien relativ klar machen muss. Also ich kann z.B. keine Objektorientierung an JavaScript erklären, weil JavaScript es sozusagen erlaubt, es falsch zu machen. Dadurch werden die Konzepte nicht so klar und können nicht so klar rübergebracht werden. Das heißt, bestimmte Konzepte müssen relativ klar unterstützt werden und so angeboten werden, dass es nicht fünf verschiedene Wege gibt, das Konzept zu umgehen und dadurch zu verfälschen.

I: [10:46] Vor allem am Anfang wissen Studenten ja noch nicht, wie man auch sauberen Code schreibt. Wie kann man die Studenten denn dazu bringen, dass sie bestimmte Richtlinien einhalten, welche zu sauberem Code führen?

B: [11:09] Erstmal sollte man möglichst früh möglichst viel Feedback geben und die Studenten auch möglichst früh sauber arbeiten lassen. Man sollte also nicht dem Gedanken verfallen: Ach, wir lassen die jetzt einfach nur machen und lassen die denken, dass es so okay ist. Sondern dass man, wenn man die machen lässt und unsauber arbeiten lässt, dann folgenden Hinweis mitgibt: Was ihr da macht ist nicht gut, aber das ist okay, weil ihr es eben noch nicht besser wisst. Man sollte die Studenten auch möglichst früh den Schmerz von schlechtem Code spüren lassen. Das hat bei uns in CE01 gut funktioniert. Da haben wir die Leute erst einmal Spaghetti Code schreiben lassen und haben diesen Code in einem extra Modul namens Clean Code nochmal aufgegriffen, um zu zeigen, wie schlecht der Code eigentlich war, den sie da geschrieben haben, weil sie nach einem halben Jahr überhaupt nichts mehr verstanden haben. Also an einigen Stellen muss man sie auch den Schmerz spüren lassen und in die Probleme reinlaufen lassen, damit sie verstehen, warum denn jetzt Clean Code zum Beispiel so wichtig ist.

I: [13:00] Du sagtest ja auch gerade, dass man möglichst früh den Leuten beibringen soll, sauberen Code zu schreiben. Kennst du da praktikable Wege, wie man das in Übungsaufgaben auch erzwingen kann?

B: [13:47] Also alles was funktional ist, kann man automatisiert z.B. über Unit Tests relativ gut einfach überprüfen. Problematisch wird es dann beispielsweise bei Variablen-

benennungen, wenn es darum geht, ob ein Name intuitiv ist oder nicht. Das kann man schlecht automatisch überprüfen. Deswegen muss man da wahrscheinlich auf Peer-Review Taktiken zurückgreifen. Der Peer bewertet dann quasi nicht den Code und nicht funktional, sondern die Verständlichkeit des Codes. Also man simuliert sozusagen das, was passieren würde, wenn jemand den Code abgibt in einer großen Firma oder ähnlichem. Man könnte auch Übungen zu Refactoring im Live Coding machen, also dass man ihnen vormacht, wie aus unleserlichem Code leserlicher Code wird.

I: [15:23] Jetzt würde ich mal in Richtung Domain Driven Design übergehen. Domain Driven Design stellt eine Herangehensweise zur Modellierung von komplexer Software dar, wobei sich dieser Ansatz stark an der zu modellierenden Fachlichkeit orientiert. Was muss man bei der Erstellung von Übungsaufgaben beachten, welche diesen Ansatz lehren sollen, obwohl diese meist ein geringes Maß an Komplexität aufweisen?

B: [16:18] Es kommt darauf an, welchen Teil von Domain Driven Design man beibringen möchte. Wenn es um die strategischen Entscheidungen geht, dann braucht man eine gewisse Grundmenge an Domäne. Die Domäne muss breit genug sein, damit man überhaupt irgendwie Teile davon identifizieren kann. Ich glaube aber die taktischen Aspekte, also z.B. Entities oder Value Objects, kann man auch beibringen, indem man in relativ kleinen Domänen unterwegs ist. Man muss nur Zeit darauf verwenden, zu iterieren und immer wieder sich darauf einlassen, Kleinigkeiten mehrfach anzumerken und dabei pedantisch zu sein. Zudem basiert DDD ja auch auf anderen Konzepten wie z.B. Objektorientierung. Diese sollte man etwas sauberer und durchdachter beibringen, weil Domain Driven Design ja eigentlich nur sehr saubere Objektorientierung ist. In der Lehre beobachte ich auch, dass Studenten irgendwie immer nur in Daten und nicht in Verhalten denken. Das ist sehr problematisch für die Objektorientierung und deswegen auch sehr problematisch für Domain Driven Design.

I: [19:19] Ich würde jetzt nochmal zu einem anderen Bereich übergehen, und zwar in Richtung Kooperation, Gruppenarbeit und Einzelarbeit. Würdest du eine Bearbeitung von Programmieraufgaben eher in Gruppen oder in Einzelarbeit bevorzugen, wenn diese dazu gedacht sind, bewertet zu werden?

B: [19:50] Generell gesprochen, auf jeden Fall in der Gruppe, weil das das einzig Realistische ist. Es gibt heutzutage kaum noch Probleme, welches man alleine sozusagen lösen kann, dazu ist es einfach zu komplex geworden. Ich möchte nicht nur einfach Einzelarbeit machen, weil ich diese dann besser bewerten kann. Ich möchte reale Dinge beibringen und

deswegen bevorzuge ich immer Gruppenarbeit. Und wenn man sozusagen einzelne Noten haben möchte, dann muss man noch einen individuellen Anteil mit einbringen. Vielleicht spricht man mit den einzelnen Personen dann nochmal über den Code und lässt sich ein bisschen was erklären oder man beobachtet das Coding. Wenn man Studierenden allerdings gerade frisch das Programmieren beibringt, dann würde ich Einzelarbeit eher sehen. Weil das muss eben jeder können und da bringt gegenseitiges Unterhalten und Kommunizieren nicht so viel. Das muss jeder für sich selber irgendwie verstehen und diesen Weg kann man auch nicht so richtig erklären. Dazu muss man sich konzentrieren und selber damit auseinandersetzen.

I: [22:46] Die Arbeitstechnik Pair-Programming beschreibt ein Verfahren, bei dem zwei Personen abwechselnd jeweils Programmieren und Feedback geben. Welche besonderen Vorteile siehst du hier in Abgrenzung zur Gruppenarbeit?

B: [22:51] Ich würde die Kombination aus Gruppenarbeit und Pair-Programming sehen. Also so wie es auch in der Realität passiert. Ich habe dann irgendwie kleinere Teams, die sich um bestimmte Bereiche kümmern. Häufig wird dann auch innerhalb dieser Gruppe nochmal Pair-Programming betrieben. Die Gruppenarbeit hat eher so etwas höhere großflächigere Vorteile, dass man sich eben Arbeit aufteilen und diese parallelisieren kann. Dazu kommt noch, dass man über bestimmte Konzepte während Planungen diskutieren kann und Lösungen aushandelt. Beim Pair-Programming geht es dann weniger um die Architektur-Ebene, sondern mehr um die Code-Ebene. Pair-Programming löst eigentlich einfach nur das Problem, dass Programmierer Menschen sind und manchmal Fehler machen und deswegen ist es eben besser, mit vier Augen anstatt mit zwei Augen zu programmieren. Außerdem kann das Abwechseln auch mal angenehm sein, wenn man gerade nicht weiterkommt.

I: [25:26] Die Lernenden brauchen ja immer eine gewisse Motivation, um überhaupt Übungsaufgaben zu bearbeiten. Um Lernenden den nötigen Druck bei der Aufgabenbearbeitung zu machen, gibt es verschiedene Ansätze. Zum einen kann Druck durch Noten und Aufgabenindividualisierung geschaffen werden und zum anderen kann ein motivierendes Lernklima schon ausreichen. Unter welchen Umständen sollte deiner Meinung nach welcher Ansatz bevorzugt werden?

B: [26:45] Ich glaube, dass die meisten Studenten anfänglich intrinsisch motiviert sind, weil sie sich höchstwahrscheinlich aus einem Grund für Informatik entschieden haben. Man muss eigentlich mehr darauf achten, diese intrinsische Motivation nicht zu zerstören.

Und ich glaube, dass es häufig der Fall ist, dass diese intrinsische Motivation zerstört wird, indem z.B. die Arbeit der Studenten nicht ernst genommen wird oder ihnen widersprüchliches Feedback gegeben wird. Ich glaube Studenten kommen dann in so eine Situation, dass sie nicht mehr motiviert sind, obwohl sie motiviert waren, weil sie merken, dass die Arbeit, die sie reinstecken, für sie nicht lohnend oder erfüllend ist. Und wenn es so weit gekommen ist, dann sind auch Lehrende ganz oft der Meinung, dass sie mithilfe von Noten, Deadlines oder Meilensteinen extrinsisch motivieren müssen. Manchmal schlägt bei den Lehrenden auch eine gewisse Arroganz durch, in dem Sinne, dass die Lehrenden ja nichts falsch machen und der Fehler bei den Studenten liegen muss. Oft wird nicht reflektiert, das man als Lehrender vielleicht selber irgendwie an der Art des Bewertens oder an der Art des Beibringens arbeiten muss, um die intrinsische Motivation zu erhalten. Wichtig dabei ist eben, dass man als Lehrender den Studenten nichts Missverständliches oder zu stark vereinfacht erzählt. Sonst läuft man Gefahr, dass wenn die Studenten Google aufmachen, dass sie dann andere Aussagen zu diesem Thema finden, die an der Sinnhaftigkeit der Vorlesung oder Übung zweifeln lassen. Wenn sie etwas richtig machen, müssen sie das auch relativ früh und eindeutig gesagt bekommen. Wenn man noch in die Richtung Constructive Alignment geht, dann muss klar sein, was geprüft wird und was verlangt wird. Um ein bisschen mehr in deine Richtung zu gehen: Man braucht kurze Feedback Zyklen, schnelles Feedback z.B. mit Unit Tests, die automatisiert passieren können. Genau dadurch ist schon ausgeschlossen, dass sie mehrere Meinungen bekommen, denn die Wahrheit liegt im Code. Entweder ist ein Test grün oder rot, da gibt es keine zwei Meinungen drüber.

I: [30:46] Wenn diese intrinsische Motivation jetzt aber doch nicht vorliegt, gibt es ja trotzdem verschiedene Möglichkeiten, wie man den Einzelnen dazu bewegt, dass er sich auch mit der Aufgabe beschäftigt und nicht einfach nur z.B. Lösungen kopiert. Eine Möglichkeit wäre natürlich, dass man Individualisierung betreibt. Was für Individualisierungsmethoden kennst du, mit denen man Aufgaben individualisieren kann, sodass eben das Kopieren von Lösungen nicht mehr so einfach ist?

B: [31:24] Was ich aus meinem Studium her kenne, sind so etwas wie Varianten A und B bei Klausuren, sodass ein Ablesen oder Abschreiben nicht möglich ist. Wenn es eher um algorithmische Belange geht, kann man relativ einfach nur die Testdaten verändern. Komplexer wird es dann, wenn man auch semantisch individualisiert, also dass man den Kontext und die Domäne ändert und dadurch individualisiert. Das ist allerdings mehr Aufwand und braucht ein Tooling. Wenn es aber darum geht, konkret zu testen, ob jemand etwas verstanden hat oder nicht, dann muss man in wirkliche Fachgespräche gehen

und nachhaken. Dies würde aber eher in Richtung Prüfungen gehen. Wir haben ja eben über Motivation gesprochen, die wahrscheinlich früher passieren sollte und nicht erst in der Prüfung. Da ist dann wahrscheinlich eher eine Individualisierung nötig oder möglich.

I: [33:28] Was wäre aus deiner Sicht wichtiger: Dass man einen sehr hohen Grad an Individualisierung hat oder dass man mehr Fokus legt auf die Geschichte, die den Aufgabenkontext bestimmt?

B: [34:29] Ich würde sagen, die Geschichte ist wichtiger. Also wenn es um Domain Driven Design geht, dann bestimmt ja die Domäne und der Kontext wirklich das, was schlussendlich im Code steht. Und wenn man nur um der Individualisierung Willen diese Geschichte und diese Domäne irgendwie unlogisch macht, dann verfehlt man sozusagen das Ziel. Man nimmt dann den Studenten wirklich die Chance, ihre Alltagserfahrungen zu nutzen, um daraus zu lernen und in Code zu überführen. Deswegen würde ich sagen, die Story darf nicht verfälscht werden.

I: [35:24] Würdest du das anders sehen, wenn man jetzt mal an Programmieraufgaben für Anfänger denkt beziehungsweise an den ersten Programmierkurs?

B: [35:42] Ich glaube, ich würde den ersten Programmierkurs relativ unabhängig von der Domäne gestalten. Deswegen würde sich mir da die Frage gar nicht stellen. Zu Beginn sind die Leute schon genug mit den grundlegenden Programmierkonzepten beschäftigt, sodass man die Domäne erst einmal außen vor lassen sollte. Bei Code and Context hat es gut funktioniert, nicht zu individualisieren, weil es jeder selber machen musste. Trotzdem sind die Studenten teilweise an einigen Problemen gescheitert und dann sind sie in den Austausch gekommen und haben sich gegenseitig geholfen. Das wäre dann ein Softskill, den sie schon beim ersten Kurs lernen. Deswegen ist es an der Stelle gar nicht so wichtig, dass das jeder selber gemacht hat, sondern einfach nur beteiligt war an der Lösungsfindung.

I: [38:32] Was wäre da deine Meinung zu abweichenden Schwierigkeitsgraden, die entstehen können, wenn man individualisiert?

B: [38:56] Ich glaube, das ist ein Punkt, auf dem man sehr stark achten muss, dass das eben nicht zu sehr auseinander läuft bei der Individualisierung. Ich glaube, das lässt sich aber manchmal nicht vermeiden, indem z.B. einfach bestimmte Kombinationen von Domänenobjekten entstehen, die nicht ganz so einleuchtend sind wie andere. An der Stelle

sollte man nochmal die Möglichkeit geben, dass da nochmal jemand drauf schaut. Dann können eben solche schwierigen Kombinationen erkannt werden und dann kann z.B. eine bessere Note gegeben werden.

I: [39:58] Wie viel mehr Aufwand würdest du persönlich investieren, um solche Aufgaben zu erstellen, die auch individualisiert werden können?

B: [40:26] Also ich kann dir keine genaue Zahl sagen, aber ich glaube, ich wäre bereit, sehr viel Arbeit reinzustecken, weil ich glaube, dass sich das auf lange Sicht lohnen wird. Also die Individualisierung wird trotzdem irgendwann passieren. Ich höre immer wieder, dass Prüfungen neu gestellt werden müssen, weil man bestimmte Aufgaben ja schon in der letzten Klausur oder im Praktikum schon hatte. Das waren immer Argumente dafür, dass eine Klausur neu geschrieben werden musste, sich neue Praktikumsaufgaben überlegt werden mussten und so weiter. Das ist eben auch eine Art der Individualisierung, nur verteilt über mehrere Semester und Jahre. Und wenn man dann einmal am Anfang relativ viel Arbeit reinsteckt, hat man sich das dann am Ende vielleicht gespart. Man muss es vielleicht noch immer überprüfen, ausbauen und verbessern, aber man schmeißt weniger weg.

I: [41:55] Ich würde jetzt mal in Richtung Automatisierung übergehen. Was muss eine Übungsaufgabe für Eigenschaften aufweisen, damit man diese auch automatisch überprüfen kann?

B: [42:22] Einfach überprüfbar sind in den meisten Fällen nur funktionale Aspekte, wenn es nur um reine Unit Tests geht. Wenn es um Architektonische Aspekte geht, wird es schon schwieriger, weil bei architektonischen Aufgabenstellungen gibt es nicht unbedingt immer eine Lösung, die richtig ist. Außer man verkauft das den Leuten so, was dann an sich schon relativ schwierig ist. Also wenn man Domain Driven Design macht, dann kann es z.B. fünf Domänenmodelle geben, die alle unterschiedlich sind. Manche sind besser, manche sind schlechter, aber grundlegend ist keine davon falsch, sondern einfach nur anders. Und da wird es problematisch mit dem Überprüfen. Man müsste sich alle Kombinationen, die halbwegs sinnvoll sind, überlegen und diese abprüfen. Das heißt, bei architektonischen Themen wird es dann schon wieder schwieriger. Außer man gibt denen sehr starke Leitplanken dahingehend, was sie machen sollen und was nicht. Das ist im ersten Moment vielleicht eine gute Empfehlung, aber wir sind im Software Design und Design heißt, es gibt nicht einfach nur eine richtige Lösung gibt. Außer man ist bei dem funktionalen, wo eben relativ klar definiert ist, 1+1 muss 2 ergeben. Also wenn man in die architektonische

Richtung gehen möchte, dann muss man diese Regeln relativ klar formulieren und nach meiner Ansicht aber auch dazu sagen, dass diese klar formulierten Regeln manchmal nur eine starke Empfehlung sind und deshalb nicht immer an jeder Stelle auch zutreffend sind. Also ich könnte mir z.B. durchaus vorstellen, dass es an einer Stelle manchmal auch eine Bidirektionale Beziehung sein darf aus bestimmten Gründen. Und wenn man das durch einen Test ausschließt, obwohl ein Student eine gute Begründung hat, dann sollte man den nicht einfach bestrafen.

I: [45:20] Hast du Ideen, wie Übungsaufgaben sowohl automatisch überprüfbar wären als auch mit einer gewissen Freiheit in der Aufgabenbearbeitung zu lösen wären?

B: [45:24] Man muss sich erst einmal selber relativ klar werden, was jetzt feste Gesetze sind, die nicht gebrochen werden dürfen und was eher Empfehlungen sind, die eventuell situationsabhängig sind. Wenn man Schichten Architektur macht, ist es z.B. eine feste Regel, dass ich Schichten beim Zugriff nicht überspringen darf. Das kann man leicht überprüfen. Anders ist es bei bidirektionalen Beziehungen im DDD, von denen meist stark abgeraten wird, aber das ist kein ultimatives Gesetz. Das heißt, ich würde sagen, man muss sich als Dozent oder auch als Softwareentwickler allgemein sehr stark überlegen, ob das jetzt etwas ist, was auf keinen Fall gebrochen werden kann oder einfach nur eine Empfehlung ist. Also man kann auch automatische Tests machen, welche ein bisschen Kreativität zulassen. Die werden eben nur unglaublich komplex zu schreiben. Also nehmen wir mal an, man überprüft, ob eine Klasse sinnvoll benannt ist. Man kann einfach überprüfen, ob die Klasse Car heißt oder man kann auch Fuzzy Überprüfung machen. Kommt z.B. irgendwo das Wort Car vor und kommt auf keinen Fall das Wort Auto vor, weil das deutsch wäre. Also man kann diese Überprüfung breiter anlegen und sozusagen Kreativität voraussehen, aber das wird unglaublich aufwendig.

I: [47:53] Ich würde jetzt nochmal zu Tools übergehen. Hast du schon mal in der praxisorientierten Lehre Tools zur Aufgabenindividualisierung angewendet und was hast du da für Erfahrungen gemacht?

B: [48:13] Nein, habe ich bisher noch nicht.

I: [48:41] Nochmal eine Frage zu automatischer Aufgabenüberprüfung mittels Tools. Inwieweit denkst du, kann Feedback von automatischen Aufgabenkorrekturen das Feedback von manuell korrigierenden Lehrenden ersetzen?

B: [49:01] Bestimmte Bereiche lassen sich gut überprüfen, also alles, was funktional ist. Teilweise lassen sich auch nicht funktionale Aspekte überprüfen. In der Algorithmik könnte man z.B. mithilfe der O-Notation das Laufzeitverhalten von verschiedenen Ansätzen überprüfen und so entscheiden, ob auf Basis der richtigen Datenstrukturen die richtige Lösungsstrategie ausgewählt wurde. Das wird dann aber auch schon wieder aufwendig, denn dann muss man irgendwie 1000 Durchläufe machen und den Mittelwert bilden und dann auch überprüfen, dass der Server nicht gerade auf Volllast ist. Sobald es aber um Design, um Architektur, um solche Sachen geht, dann sollte man auch mal jemand menschlichen drüber schauen lassen, denn ich glaube, das ist nicht immer komplett automatisiert überprüfbar.

I: [51:10] Wenn man Studenten Übungsaufgaben geben will, dann muss man diese auf irgendeine Art verteilen und vielleicht auch Lösungen einsammeln. Mit welchen Plattformen hast du Erfahrungen gemacht, wo sich eben das realisieren lässt?

B: [51:43] Also für Abgaben ist glaube ich Git das Beste, denn es ist versioniert und wenn man es richtig einstellt, dann weiß man auf jeden Fall nicht nur, wer es am Ende rechtzeitig hochgeladen hat, sondern man kennt auch die Zwischenstände. Man weiß also, wie der Student zur Lösung gekommen ist. Bei Mathematikklausuren schreibt man ja auch nicht nur die Lösung hin, sondern auch den Weg. Verteilen kann man auch über Git, da kommt es aber auf die Art der Aufgaben an. Mit Unit Tests versehene Aufgaben würde ich über Git verteilen, aber wenn es z.B. um komplexere Domänen geht, werden auch andere (visuelle) Darstellungsformen nötig, die auf einer Website oder in einem Wiki besser aufgehoben sind.

I: [52:45] Dann nochmal ein letztes Thema, und zwar Feedback und Beratung. Du hast ja auch mehrmals erwähnt, dass Feedback nach wie vor wichtig ist, auch wenn man automatische Mechanismen mit integriert. Was hältst du von modernen Messengern als Kommunikationsplattform zwischen Lehrenden und Lernenden?

B: [53:23] Gut und wichtig, weil es die Feedback Zyklen kleiner macht. Dazu kommt noch, dass sich bestimmte Arten der Kommunikation für verschiedene Themen besser eignen als für andere. Man kann z.B. Feedback über E-Mail geben, das dauert allerdings relativ lange, sodass die Kommunikation und das Feedback einfach nicht so direkt sind. Wenn man diese Richtung betrachtet, gibt es eine relativ klare Hierarchie, welche Kommunikationsart sich eignet, wenn man bestimmte Feedback Geschwindigkeiten haben will. Das Beste ist face to face Kommunikation, vor dem Code sitzen und mit dem Studenten zu-

sammen durchgehen, was ist gut und was ist falsch. Gerade jetzt funktioniert das nicht. Und die nächstbessere Möglichkeit ist sozusagen dann im Videochat, weil dann eben auch Mimik dabei ist. Dann kommt die Möglichkeit von irgendetwas Verbalem, also direkte Kommunikation verbal über Voice Chats über z.B. Zoom oder Discord. Die nächste Möglichkeit ist dann sozusagen Instant Messaging, also textuell, aber dann sehr direkt, sodass direkt und schnelle geantwortet wird. Das ist auf jeden Fall besser als Kommunikation über E-Mail.

Eidesstaatliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum: Köln, 17. April 2021

Rechtsverbindliche Unterschrift: