INAUGURAL - DISSERTATION

zur
Erlangung der Doktorwürde
der
Naturwissenschaftlich-Mathematischen Gesamtfakultät
der
Ruprecht-Karls-Universität
Heidelberg

vorgelegt von
Diplom-Ingenieur Walter Kicherer
aus Reutlingen

Tag der mündlichen Prüfung: 14. Juli 2008

Objektorientierter Softwareentwurf in der Sekundarstufe II

Gutachter: Prof. Dr. Gabriel Wittum
Prof. Dr. Rosemarie Boenicke

Zusammenfassung

Der objektorientierte Softwareentwurf spielt in der Bildung eine immer wichtigere Rolle. Mit ihm kann die Modellbildungskompetenz und die Problemlösefähigkeit, wie sie u.a. in der Pisa-Studie [Bm01] gefordert werden, gut gefördert werden. In dieser Arbeit wird ein Konzept hierfür entwickelt, umgesetzt und evaluiert.

Als Leitgedanken bei der Entwicklung dient die Förderung der Modellbildungskompetenz und der Problemlösefähigkeit, wobei der objektorientierte Softwareentwurf in seiner ganzen Breite von der Anforderungsanalyse über das Design bis zur Implementierung bearbeitet wird.

Dazu werden zuerst die fachlichen Inhalte strukturiert und diese anschließend zusammen mit einer Lernzieltaxonomie in einer Lehrzielmatrix verknüpft, um so die Lernziele festzulegen. Als didaktisches Konzept wird ein dem üblichen Entwicklungsverlauf beim Softwareentwurf entgegengesetztes Vorgehen gewählt und begründet. Die entwickelten Lernprozesse werden dann mit Hilfe von Aktivitätsdiagrammen dargestellt und feiner ausgearbeitet.

Zur Evaluation des Konzepts wurden die ausgearbeiteten Einheiten an 3 Schulen umgesetzt und mit einer Kontrollgruppe an 3 weiteren Schulen verglichen. Dazu wurde ein Test entwickelt und zusammen mit einem IQ-Test und Fragebögen zu den Randbedingungen von den Schülern und Eltern ausgefüllt.

Das Ergebnis der Evaluation zeigt, dass die Verwendung dieses Konzepts aber auch der Einfluss des Lehrers und die Beschäftigung der Schüler mit dem Programmieren in der Freizeit wichtige Faktoren sind. Das entwickelte Konzept ist gut geeignet in den objektorientierten Softwareentwurf einzuführen und die Problemlösefähigkeit und die Modellbildungskompetenz im Vergleich zur Kontrollgruppe zu fördern.

Abstract

Object-oriented software engineering becomes more and more important in education. Modelling competence as well as problem-solving skills, as postulated by the PISA study [Bm01], can be well improved by it. In this thesis a concept for it will be developed, implemented and evaluated. The guiding ideas behind are the enhancement of both, the modelling competence and the skills of problem-solving, while object-oriented software engineering will be covered in its entire spectrum from requirement analysis through design up to implementation.

The technical subjects will be structured first. Then they will be linked to the taxonomy of learning objectives for formulating educational goals. As didactic concept a procedure will be chosen and justified which runs contrary to the common way of software engineering. The resulting learning processes will then be presented by means of the well known UML activity diagrams and further elaborated.

These final units are then implemented at three academies and compared against a control group at three other academies. For this purpose a test was designed. It was filled out by the students together with an IQ test and questionaires about basic conditions.

The outcome of this evaluation shows that the implementation of this concept as well as the influence of the teacher and programming by the students during their free time are important factors. The newly developed concept is highly suitable as an introduction to object-oriented software engineering and is improving modelling competence as well as problem-solving skills.

II

Einen jungen Menschen unterrichten heißt nicht,
einen Eimer zu füllen,
sondern ein Feuer anzuzünden.

Aristoteles

Mein Dank gilt
Lisi für Ihre Geduld und Liebe
als ich klein war und
Connie für all dies seit ich groß bin.

Weiterhin möchte ich Eugen danken,
da Ihm meine Bildung immer
sehr am Herzen lag.

Jonas und Hanna danke ich,
da ich durch diese Arbeit
nicht so viel Zeit für sie hatte
wie ich mir wünschte.

Inhaltsverzeichnis

1 Einleitung

Spätestens mit dem Erscheinen der *Empfehlung für ein Gesamtkonzept zur informatischen Bildung an allgemein bildenden Schulen* der Gesellschaft für Informatik e.V. [GI00] ist klar, dass Informatik auch im Unterricht eine wichtige Rolle spielt. Darin werden als Leitlinien die

- *Interaktion mit Informatiksystemen,*
- *Wirkprinzipien von Informatiksystemen,*
- *informatische Modellierung,*
- *Wechselwirkungen zwischen Informatiksystemen, Individuum und Gesellschaft*

erwähnt.

Die Informatik wird immer stärker als ein allgemein bildendes Fach angesehen. P. Hubwieser konstatiert, dass die informatische Bildung auf zukünftige Lebenssituationen vorbereitet, indem sie sich bemüht, das Streben der Menschen zur Automatisierung von geistigen Tätigkeiten zu verdeutlichen und damit auch deren Grenzen aufzeigt [Hu01, S. 62f]. Dies ist besonders wichtig, um auch die gesellschaftlichen Auswirkungen der Informatik zu verstehen und einzuschätzen, wie es in der oben genannten Empfehlung zum Punkt *Wechselwirkung von Informatiksystemen, Individuum und Gesellschaftlichem* gefordert wird.

Der objektorientierte Softwareentwurf, welcher in dieser Arbeit im Mittelpunkt steht, ist eine noch relativ junge Disziplin der Informatik. Sie stellt Verfahren und Werkzeuge zur Verfügung, um Software zu entwerfen. Analog zur Betrachtung unserer Umwelt werden dort die Teile der Software als Objekte betrachtet. Objekte mit denselben Eigenschaften, aber unterschiedlichen Ausprägungen dieser Eigenschaften werden in Klassen zusammengefasst. So haben z.B. Autos unterschiedliche Farben, Gewichte und Motorleistungen, sie bilden die Klasse *Auto*. Ein konkretes Objekt dieser Klasse hat die Farbe blau, das Gewicht 860 kg und eine Leistung von 120 kW.

Wie lassen sich nun die von der Gesellschaft für Informatik aufgestellten Leitlinien mit Hilfe des objektorientierten Softwareentwurfs umsetzen?

- Die erste Leitlinie, die Interaktion mit informatischen Systemen, kann durch Bedienung der entsprechenden Software bei der Entwicklung (Case-Tools, Textverarbeitung zur Dokumentation, ...), vor allem jedoch beim Entwurf von graphischen Oberflächen, in dem also die Schnittstelle zur Interaktion Mensch-Maschine erstellt wird, umgesetzt werden.
- Die zweite Leitlinie, die Wirkprinzipien von Informatiksystemen, lässt sich exemplarisch mit Hilfe des objektorientierten Softwareentwurfs verdeutlichen. Nach dem Entwurf und der Realisierung eines Programms durch Lernende sind die inneren Zusammenhänge und je nach Programmgröße auch die Komplexität der Systeme klarer.
- Die letzte Leitlinie, die Wechselwirkung zwischen Informatiksystemen, Individuum und Gesellschaft ergibt sich indirekt über die Beschäftigung mit dem Thema.
- Der dritte Leitlinie, die informatische Modellierung, soll etwas genauer betrachtet werden. Zuerst muss allerdings geklärt werden, was unter Modellierung und ganz speziell unter informatischer Modellierung verstanden wird.

P. Hubwieser [Hu99] definiert den Begriff Modellieren

> *... als eine abstrahierte Beschreibung eines realen oder geplanten Systems, das die für eine bestimmte Zielsetzung wesentlichen Eigenschaften dieses Systems enthält. Die Erstellung einer solchen Beschreibung nennen wir Modellbildung oder Modellieren.*

Er postuliert dann das Modellieren als den roten Faden im Unterricht und fordert, die Modellierungstechniken einzeln vorzustellen, auf konkrete Probleme anzuwenden und die Modelle umgehend zu implementieren. Eine dieser Modellierungstechniken ist bei ihm die objektorientierte Modellierung.
Die Gesellschaft für Informatik versteht unter informatischem Modellieren, dass ein Modell

> *.. im wesentlichen die Abgrenzung eines für den jeweiligen Zweck relevanten Ausschnitts der Erfahrungswelt, die Herausarbeitung seiner wichtigen Merkmale unter Vernachlässigung der unwichtigen sowie seine Beschreibung und Strukturierung mit Hilfe spezieller Techniken aus der Informatik ..*

ist [GI00].
Eine dieser in der Informatik verwendeten Techniken ist die Beschreibung der Modelle mittels der Unified Modeling Language (UML [OMG00]), welche eine Zusammenführung u.a. der Arbeiten von Booch, Rumbaugh und Jacobsen darstellt [BRJ98]. Der objektorientierte Softwareentwurf stellt nun Verfahren zur Verfügung, um Modelle im obigen Sinne zu erstellen. Als Darstellungsform dienen dabei die Diagramme der UML.

Die Bedeutung der Problemlösefähigkeit und des Modellierens stellt wiederum P. Hubwieser in unter dem Aspekt der methodischen Prinzipien vor. Er verwendet dabei die Definition des Begriffs 'Problem' nach Edelmann:

> *Ein Problem ist also durch drei Komponenten gekennzeichnet:*
> - *unerwünschter Anfangszustand,*
> - *erwünschter Zielzustand,*
> - *Barriere, die die Überführung des Anfangszustandes in den Zielzustand im Augenblick verhindert.* (zitiert nach [Hu01, S. 68])

Er erwähnt, dass die Problemlösefähigkeit eine anerkannte Leitlinie der informatischen Bildung ist und die Modellbildung und Simulation das Prinzip der Unterrichtsgestaltung sein sollte. Weiterhin stellt er fest, dass Probleme, welche sich durch Anwenden von Strategien oder durch Systemdenken lösen lassen, optimal wären.

Es zeigt sich also, dass der objektorientierte Softwareentwurf gut geeignet ist, wichtige informatische Inhalte exemplarisch zu vermitteln. Er ist in der Zwischenzeit so bedeutend, dass er auch explizit in Lehrpläne aufgenommen wurde (z.B. [It01]).
Die allgemeine Wichtigkeit, dass die Modellbildungskompetenz und Problemlösefähigkeit gefördert werden sollen, hebt auch die Pisa-Studie hervor, indem dort erwähnt wird, dass

> *... in der internationalen Pisa-Rahmenkonzeption ... die Fähigkeiten zur Vernetzung und Modellierung als Ziele des Mathematikunterrichts im Vordergrund* [Bm01, S. 25]

stehen. Dies lässt sich sicher auch auf den Fachbereich der Informatik übertragen, da beide Gebiete eng miteinander verwandt sind.

Es ist aber festzustellen, dass bisher nur wenig didaktisch fundierte Konzepte in der Informatik zur Umsetzung dieser Thematik zur Verfügung stehen. Meist sind es entweder nur Konzepte zu einzelnen Unterrichtsstunden bzw. kurzen Unterrichtseinheiten oder der Schwerpunkt liegt auf der Programmierung bzw. genauer auf der Vermittlung der Syntax der Programmiersprache. Vereinzelt existieren Konzepte, bei welchen der Schwerpunkt fast ausschließlich auf der Modellierung liegt. Diese vernachlässigen dabei die Codierung und damit das Testen des Modells. Des Weiteren wurde keines dieser Konzepte evaluiert und so auf seine Wirksamkeit hin überprüft. Detailliert wird auf die bisher bestehenden Konzepte im nächsten Kapitel eingegangen.

Aus diesem Dilemma heraus entstand die Idee, ein durchgängiges Unterrichtskonzept zum Thema *Objektorientierter Softwareentwurf* zu entwerfen. Es soll ein in Bezug auf die Bildungsstufe (Sekundarstufe II, Tertiärbereich) flexibles Konzept erstellt werden, welches dann jedoch konkret in der Sekundarstufe II im Rahmen des Lehrplans des technischen Gymnasiums, Profil Informationstechnik in Baden-Württemberg [It01] umgesetzt wird . Um die Flexibilität zu erhalten müssen jedoch, abhängig von der Bildungsstufe, die Inhalte und deren Tiefe (Lernzieltaxonomie) angepasst werden.

Aus dem oben Genannten ergibt sich, dass dies eine interdisziplinäre Arbeit darstellt. Sie verbindet Inhalte der Informatik, genauer aus dem Bereich des objektorientierten Softwareentwurfs und der Programmierung, mit denen der Pädagogik. Sie verbindet also die technisch-mathematischen mit den Geistes- und Sozialwissenschaften.
Im Vordergrund der Forschung steht die Entwicklung eines didaktisch-methodischen Konzepts zur Vermittlung informatischer Inhalte und dessen Evaluation.
Diese Arbeit stellt einen Beitrag zur Didaktik der Informatik dar, einerseits für die Theorie der Didaktik der Informatik bei der Entwicklung des Konzepts, andererseits im empirischen Sinne bei der Evaluation des Konzepts.
Das Konzept wird in seiner Grobform für die Sekundarstufe II, aber auch für den tertiären Bereich entwickelt. Die Feinform und die Evaluation der Umsetzung erfolgt jedoch nur in der Sekundarstufe II.
Im Rahmen der Evaluation des Unterrichtskonzepts ergeben sich des Weiteren Bezugspunkte zur Statistik, um die Erhebungsdaten auszuwerten. Die statistischen Verfahren werden jedoch nur als Instrument benutzt, ohne in diesem Bereich neue Erkenntnisse zu suchen.

Die Arbeit gliedert sich in drei Blöcke und einer abschließenden Diskussion. Nach dieser Einleitung werden in Kapitel 2 bestehende Konzepte untersucht. Dabei wird in einer Literaturrecherche einerseits die fachwissenschaftliche Literatur untersucht, bei welcher der objektorientierte Softwareentwurf und nicht die Didaktik im Vordergrund steht und andererseits die fachdidaktische Literatur, bei welcher die Vermittlung der Inhalte im Vordergrund steht.

Anschließend wird in Kapitel 3 das Unterrichtskonzept entworfen. Nach Verdeutlichung der Leitgedanken, also der Ziele des Konzepts, und den spezifischen Anforderungen der Bildungseinrichtungen werden die fachwissenschaftlichen Inhalte herausgearbeitet und danach aus fachdidaktischer Sicht gewichtet. Nun kann das grobe Konzept mit Hilfe informationstechnischer Darstellungsweisen und das detailliertere Konzept entworfen werden.
Kapitel 4 ist der Umsetzung und Evaluation des Unterrichtskonzepts gewidmet. Nach Vorstellung der forschungsleitenden Hypothesen zur Evaluation und der Messwerkzeuge erfolgt die Darstellung der Evaluationsergebnisse.
Zum Schluss erfolgt in Kapitel 5 eine Wertung des Unterrichtskonzepts unter dem Ergebnis der Evaluation.
Im Anhang finden sich die konkreten Unterrichtseinheiten inkl. aller Arbeitsblätter und Lösungen, wie sie an die teilnehmenden Lehrer der Umsetzungsgruppe versandt wurden. Auch die Evaluationswerkzeuge in Form der verwendeten Fragebögen sind im Anhang zu finden, inkl. einer Musterlösung und eines Bewertungsschemas des Vor- und Nachtests. Des Weiteren findet sich im Anhang eine Übersicht der erstellten Software, welche einen zentralen Teil dieses Unterrichtskonzepts darstellt.

2 Bestehende didaktische Konzepte

2.1 Bestehende Konzepte in der fachwissenschaftlichen Literatur

Die meiste Literatur zum objektorientierten Softwareentwurf stammt aus dem akademischem Umfeld und hat als Adressaten einerseits Studenten und andererseits akademisch Gebildete. Die Auswahl[1], welche aufgrund der in der Zwischenzeit großen Anzahl von Veröffentlichungen nur exemplarisch sein kann, bezieht sich auf die Jahre 2001/2002.

Meist lässt sie sich in 3 Kategorien einteilen:

1. programmiersprachen- bzw. codeorientierter Inhalt,
2. modellierungsorientierter Inhalt und
3. managementorientierter Inhalt.

In der ersten Kategorie wird meist nicht bzw. nur kurz über die Modellierung gesprochen. Im Vordergrund steht die Programmiersprache selbst oder die Grundkonzepte der objektorientierten Programmierung wie z.B. die Unterscheidung von Klasse und Objekt, Beziehungen und Vererbung oder das *Information Hiding*[2]. Diese Grundbegriffe werden dann meist an einem kleinen praktischen Beispiel erläutert und anhand der gewählten Sprache verdeutlicht. Beispiele für diese Kategorie sind [Ie00], [Br99], [Mi02], [Jo02].
Zum Teil wird jedoch in dieser Kategorie der Modellierung auch etwas mehr Platz eingeräumt. Ein Beispiel hierfür ist *Thinking in C++* [Ec00a] bzw. *Thinking in Java* [Ec00b] von Bruce Eckel. In einem Kapitel werden die allgemeinen Eigenschaften der objektorientierten Programmierung wie z.B. Klasse und Objekt, Beziehungen und die Vererbung anhand von UML-Diagrammen erläutert und kurz auf das Thema Analyse und Design eingegangen indem eine 5-schrittige Vorgehensweise vorgeschlagen wird.
Der eigentliche Teil des Buchs bezieht sich dann jedoch auch auf die objektorientierte Programmierung und dabei auf die Besonderheiten der Sprache C++ bzw. Java.

In der zweiten Kategorie wird die Modellierung betont. Dabei wird sehr allgemein und unabhängig von einer Programmiersprache auf das methodische Vorgehen und oft auch auf die Formalien der UML eingegangen. Ein Buch dieser Kategorie ist *Applying UML and Patterns* von Craig Larman [La02]. Hier, wie in allen Büchern dieser Kategorie, werden ausgehend von der Reihenfolge beim realen Softwareentwurf die einzelnen methodischen Schritte von der Analyse zum Design durchgeführt. So wird zuerst geklärt, welche Anforderungen die Software zu lösen hat, dann werden die entsprechenden Methoden wie z.B. die Use Cases eingeführt. Danach kommt die genauere Definition im Rahmen einer Analyse. Anschließend wird im Design die Softwarearchitektur festgelegt. Zuletzt erfolgt die Codierung und der Test.
In diesen Büchern wird die Codierung nur sehr kurz angesprochen, Codefragmente sind fast nicht bzw. gar nicht zu finden. Der Schwerpunkt liegt darin, wie Klassen, Vererbungen und Beziehungen zu finden sind und wie ein gutes Modell aussieht.
In diese Kategorie gehören auch die Bücher von Booch [Bc94], Rumbaugh [Ru91] und Jacobsen [Ja92], welche als die Väter der Unified Modeling Language gelten. Dabei besit-

1 Nicht berücksichtigt sind Werke, welche sich inhaltlich nicht nur dem objektorientierten Softwareentwurf widmen. Meist lassen sich jedoch in diesen die einzelnen objektorientierten Kapitel auch in eine der genannten 3 Kategorien einteilen.

2 Begriff: siehe Glosar (Anhang E)

zen aber alle 3 Bücher auch Teile, die mehr managementorientiert sind, also bereits in die nächste Kategorie gehören. Dies gilt vor allem für [Ja92]. Dies zeigt bereits den Weg der Drei hin zum gemeinsam entwickelten *Unified Process* [BRJ01].
Weitere Beispiele der zweiten Kategorie sind [Ne02], [Oe98], [Ka01]. Zum Teil werden jedoch nur die Notationen und Möglichkeiten der Unified Modeling Language vorgestellt, ohne allzu tief in die Methodik der Klassifizierung einzugehen (z.B. [Fo00]).

Noch weiter gehen die Werke von P. Kruchten zum *Unified Process* [Kr99], K. Beck zum *Extreme Programming* [Be00] oder G. Versteegen zum V-*Modell* [Ve00], welche nicht auf das Finden von Klassen bzw. gutes Design, sondern eher auf den Ablauf des Entwicklungsprozesses, auf die dabei zu entstehenden Dokumentationen und den betriebsinternen Ablauf abzielen. Sie sind der dritten Kategorie zuzuordnen. Hier steht nicht mehr das Modell, sondern der betriebsinterne Ablauf im Vordergrund. In diesen Werken ist kein Programmcode und auch fast keine Modelldarstellung von Software zu finden. Schwerpunkt ist das Management von Softwareprojekten, also wie die Ressourcen Zeit, Geld und Personal organisiert werden müssen, um zu einem optimalen Ergebnis zu kommen.
Zum Teil können diese Verfahren auch in nicht-objektorientierten Softwareprojekten eingesetzt werden.

Eine Ausnahme bei der Einteilung in die 3 Kategorien programmiersprachen- bzw. codeorientiert, modellierungsorientiert und managementorientiert bildet [Ba99]. Dort werden zuerst die Grundkonzepte mit Hilfe von UML-Diagrammen erläutert und anschließend die Umsetzung in Java. Dieses Vorgehen wird jeweils gesondert für alle Grundkonzepte angewandt, also für Objekte, Klassen und ihre Eigenschaften, zuerst allgemein in Form von Objekt- bzw. Klassendiagrammen und anschließend in der Programmiersprache Java. Danach werden Ereignisse in Form von Sequenzdiagrammen behandelt und wiederum anschließend in Java realisiert.
Auf diese Weise werden Vererbung, aber auch Kontrollstrukturen und allgemeine Probleme der Informatik, wie z.B. das Sortieren und die Effizienz von Algorithmen angesprochen.
Im Rahmen dieses Aufbaus kommt auch die Vermittlung von Modellierungsmethoden nicht zu kurz, welche dann an vielen Beispielen genauer erläutert werden. Die letzte Kategorie, die managementorientierten Inhalte, kommen jedoch weniger zum Tragen.
Am Ende des Buchs erfolgt dann eine Einführung in C++.

Bei den Managementverfahren (3. Kategorie), aber auch bei den modellierungsorientierten Verfahren (2. Kategorie) wird der Ablauf beim Softwareentwurf deutlich. Da der Ablauf oft auch als didaktisches Konzept verwendet wird, soll dieser hier an einem ausgesuchten Verfahren, dem Unified-Process [Kr99], vorgestellt werden.
Zeitlich wird bei diesem der Softwareentwurf in 4 Phasen eingeteilt (siehe Abbildung 1): Konzept, Entwurf, Konstruktion und Übergang. Dies sind jedoch nur grobe Phasen, welche weiter in kleinere Stücke, die Iterationen, unterteilt werden. Am Ende einer Iteration, beim Meilenstein, ist genau beschrieben, was erreicht sein muss. Die Anzahl der Iterationen hängt von der Größe des Softwareprojekts ab.

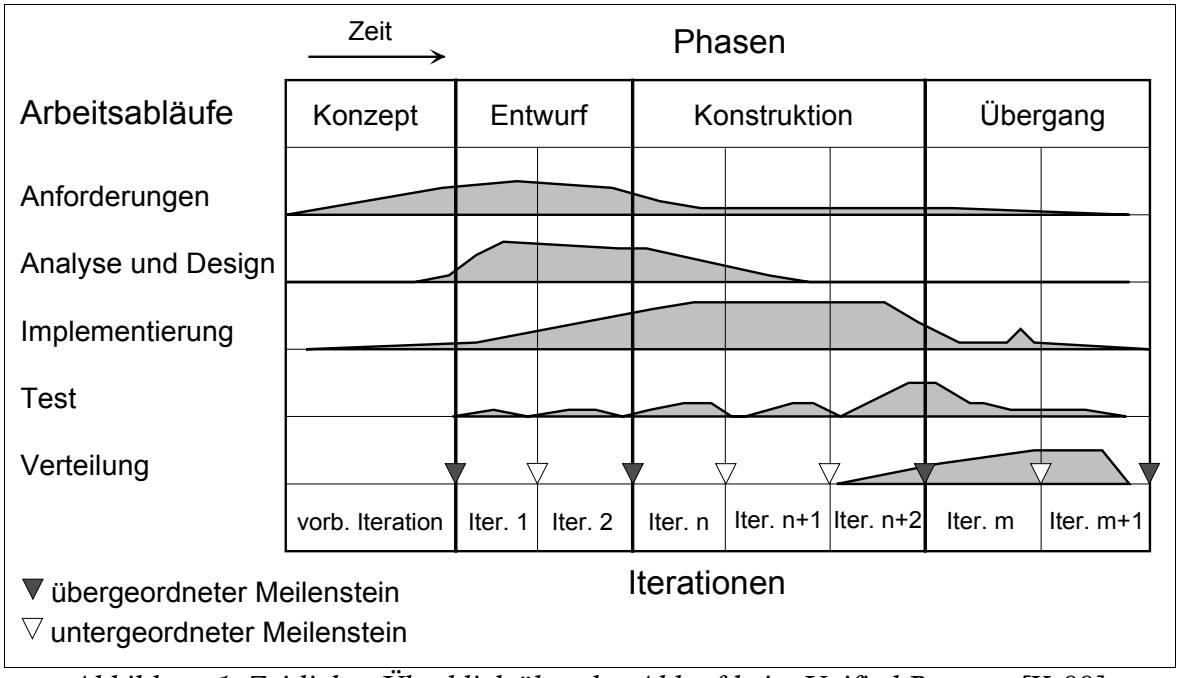


Abbildung 1. Zeitlicher Überblick über den Ablauf beim Unified Process [Kr99]

Orthogonal zu den Phasen und Iterationen sind die 5 Arbeitsabläufe Anforderungen, Analyse und Design, Implementierung, Test und Verteilung zu finden. Sie stellen die Abläufe (engl.: work flow) dar, welche beim gesamten Entwurf innerhalb einer Iteration verwendet werden. Dabei wird festgelegt, welcher Arbeiter (engl.: worker) welche Tätigkeiten beim Softwareentwurf erledigen muss, und wie diese dokumentiert werden müssen.
Im Grunde werden innerhalb jeder Iteration fast alle 5 Arbeitsabläufe durchlaufen. Allerdings sind die Arbeitsabläufe – wie in Abbildung 1 zu erkennen – in den einzelnen Phasen bzw. Iterationen unterschiedlich gewichtet, d.h. sie verursachen unterschiedlich viel Aufwand. So ist z.B. in der Entwurfsphase der Arbeitsablauf *Anforderungen* sehr stark vertreten, der Ablauf *Test* hingegen sehr schwach. Meist sind in allen Iterationen die meisten Arbeitsabläufe enthalten. Nach Erreichen eines Meilensteins beginnen wieder alle Arbeitsabläufe von vorne, um so iterativ der Lösung näher zu kommen.
Die genauere Betrachtung des zeitlichen Ablaufs beim Unified Process zeigt exemplarisch auch den oben erwähnten didaktischen Ablauf bei der Vermittlung der objektorientierten Inhalte in der fachwissenschaftlichen Literatur. Dabei wird zuerst auf die Methoden der Konzeptbildung eingegangen (z.B. über Use Cases), dann auf die Methoden des Entwurfs und dann auf die der Konstruktion[3].

3 Die Phase *Übergang* und der Arbeitsablauf *Verteilung* sollen hier nicht betrachtet werden.

Entgegen dem Unified-Process sind dabei methodisch die Phasen eng mit den Arbeitsabläufen verknüpft. So wird in der Konzept-Phase meist nur der Arbeitsablauf *Anforderungen* beschrieben, in der Entwurfs-Phase der Arbeitsablauf *Analyse und Design* und in der Konstruktions-Phase die Arbeitsabläufe I*mplementierung* und *Test*.
Der Ablauf innerhalb einer Iteration lässt sich stark vereinfacht mit Hilfe eines Ablaufplans darstellen (siehe Abbildung 2). Dabei ist allerdings die Gewichtung in den einzelnen Iterationen nicht darstellbar.
Es sind wieder die Abläufe *Anforderung*, *Analyse und Design*, *Implementierung* (Programmierung) und *Test* zu erkennen, wie sie nicht nur bei der Bearbeitung von Projekten, sondern – wie bereits oben erwähnt – auch als didaktische Reihenfolge zur Vermittlung der objektorientierten Inhalte in der fachwissenschaftlichen Literatur verwendet werden.

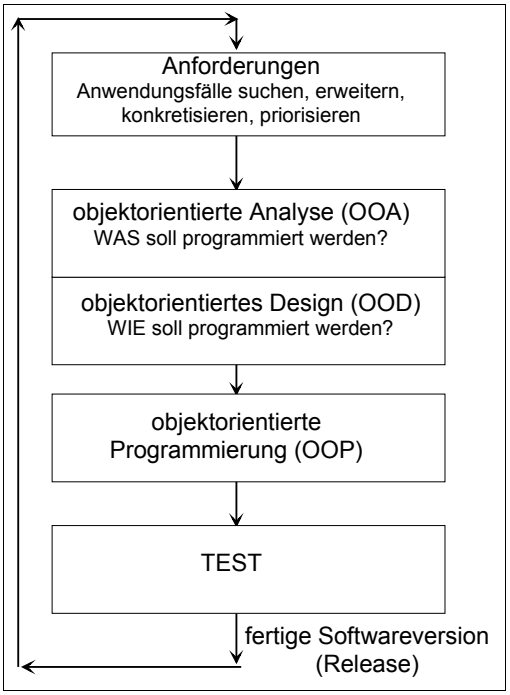


Abbildung 2. Der Ablauf innerhalb einer Iteration

2.2 Bestehende Konzepte in der fachdidaktischen Literatur

Im ersten Kapitel wurde bereits erwähnt, dass es kaum durchgängige, fundierte Konzepte für die Vermittlung der Objektorientierung gibt. Erst recht sind diese nicht evaluiert. Hier sollen exemplarisch einige Konzepte kurz vorgestellt werden, welche in der fachdidaktischen Literatur zu finden sind.

Ein recht interessantes Konzept wurde auf der 8. Fachtagung Informatik und Schule (INFOS99) der Gesellschaft für Informatik von J. Magenheim et all [Ma99] vorgestellt. Nach einer Ermittlung der Anforderungen über Anwendungsfälle und einem Analyse-Modell mit CRC Karten (nach [Be89]) wird das Designmodell als Klassendiagramm erstellt. Der Übergang zur Programmierung erfolgt dann über eine Dekonstruktion. Dabei wird zum selben Problem ein bestehendes Java-Programm vorgestellt, allerdings nicht unbedingt passend zum bisher erarbeiteten Klassenmodell. Es wird also die Lösung eines fremden Entwicklers vorgestellt und als Musterlösung verwendet. Dadurch können die Lernenden die Funktionalität der Software erkunden und eine partielle Analyse der Implementierung vornehmen.
Auf derselben Tagung wurde auch eine Umsetzung der objektorientierten Programmierung in der Schule vorgestellt [Fü99]. Nach kurzer und allgemeiner Vorstellung der Objektorientierung erfolgt eine Diskussion über mögliche Programmiersprachen. Im eigentlichen Kon-

zept wird mit den Schülern[4] ein Memory-Spiel entworfen. Dabei liegt der Schwerpunkt auf dem Finden von Klassen und deren Eigenschaften. Der Autor sieht als Resümee die Umsetzung der Objektorientierung im Unterricht sehr skeptisch.
Auf der nächsten Fachtagung (Infos 2001) wurde ein Konzept zur Umsetzung mit Hilfe von Lego Mindstorms Robotern vorgestellt [Di01]. Nach Klärung der Grundkonzepte erfolgt auch hier der Umstieg zur Programmierung über die Dekonstruktion. Danach wird die Vererbung über einen nach links bzw. im Quadrat fahrenden Roboter eingeführt. Dies ist sehr unbefriedigend, da dies zusätzliche Fähigkeiten eines Roberts sind und deshalb als zusätzliche Methoden verwirklicht werden sollten. Der Sinn der Vererbung bleibt hier auf der Strecke.
Weiterhin stellte bei dieser Tagung L. Humbert ein Modulkonzept für die Sekundarstufe II vor [Hb01]. Darin nimmt auch die objektorientierte Modellierung einen Platz ein. Er erwähnt, dass ein zunehmender Fundus von Problemstellungen hierzu vorliegt, ohne dies jedoch zu belegen. Ein Konzept für dieses Modul wird nicht vorgestellt.

In einem in der Einleitung bereits erwähnten Buch von P. Hubwieser geht der Autor auf die Diskussion Programmieren kontra Modellieren ein [Hu01, S. 87ff][5]. Er kommt dabei zum Schluss, dass die Modellierung zwar im Vordergrund stehen, aber die Programmierung im Sinne der Codierung bzw. Implementierung auch ein Teil des Unterrichts darstellen sollte. In diesem Werk wird aber nicht nur das Thema *objektorientierte Modellierung* theoretisch abgehandelt, sondern auch ein Konzept für den Unterricht vorgestellt: Es soll eine Simulation für eine kleine Eisenbahnanlage entwickelt werden ([Hu01], S. 201ff). Hierbei wird, wie wir bereits in der objektorientierten Fachliteratur gesehen haben, ein Entwicklungsprozess nachgeahmt. Die didaktischen Schritte werden also in der Reihenfolge *Anforderungen ermitteln*, *Objektmodell*, *Klassenmodell mit Vererbung* und zum Schluss die *Implementierung* durchgeführt. Die Klasse *Zug* wird mit Hilfe eines Zustandsdiagramms während der Erstellung des Klassendiagramms genauer beschrieben.
Anschließend wird der sequentielle Ablauf über ein Sequenzdiagramm dargestellt und die Problematik der Nebenläufigkeit über die Verwendung mehrerer unabhängiger Züge eingeführt.

Für die Erkundung bestehender Konzepte im schulischen Umfeld ist eine kurze historische Entwicklung der objektorientierten Didaktik in der Literatur wie sie z.B. in der Fachpublikation *Log In* zu finden ist recht interessant. Diese soll im Folgenden zusammengefasst dargestellt werden. Verfolgt werden soll die Zeit zwischen den Jahren 1996 und 2002. In dieser Zeit erkennt man wie der Schwerpunkt von der Codierung zur Modellierung verlagert wird. Dabei bleibt aber in den Konzepten die Modellierung meist nur ein Wunsch, ohne wirklich ein großes Gewicht im Vergleich zur Programmierung zu bekommen.

Im Jahr 1996 erschien eine zweiteilige Veröffentlichung eines didaktischen Konzepts zur Objektorientierung [He96]. Darin wird auf die Grundkonzepte der objektorientierten Pro-

4 Da fast ausschließlich männliche Schüler an der Evaluation beteiligt waren und um das Leseverständnis zu verbessern, wird in dieser Arbeit nur der männliche Begriff verwendet. Hiermit sind jedoch auch die weiblichen Teilnehmerinnen gemeint. Es zeigt sich jedoch auch, dass leider zu wenig Schülerinnen an der Informatik interessiert sind.

5 Dort findet sich auch eine ausführliche Diskussion mit vielen Zitaten.

grammierung (Objekt, Klasse, Vererbung, Polymorphie) eingegangen. Die Programmiersprache selbst (Oberon) nimmt jedoch einen sehr großen Raum ein.

Geht man ein Jahr weiter, so wird unter *objektorientierter Programmierung* die Programmierung einer graphischen Oberfläche verstanden [Tu97]. Die *Objektorientierung* bezieht sich dabei primär auf die Verwendung vorgegebener Klassen. Als Programmierparadigma wird weiterhin strukturiert programmiert, wobei nun die klassischen Ein- und Ausgabefunktionen durch Objekte bzw. statische Klassen der gegebenen Klassenbibliothek ersetzt werden.
Im Jahr 1997 wird dann in einem Artikel beschrieben, wie man mit Java strukturiert und objektorientiert programmieren kann [Ba97]. Allerdings stehen in einer kurzen Aufzählung nur die Eigenschaften der Objektorientierung, wieder mit dem Schwerpunk auf den Grundkonzepten. Es besteht jedoch die Forderung, sich diesem Thema mehr zu widmen.
Im nächsten Jahr, 1998, gelangt dann das Thema in die Sekundarstufe I. In einem Beitrag [Kn98] wird über ein Unterrichtskonzept gesprochen, welches über Standardanwendungen wie z.B. einer Textverarbeitung die Begriffe *Objekt*, *Attribut* und *Methode* vermittelt.
Im selben Jahr werden in einem anderen Beitrag [vl98] wiederum die Grundkonzepte der objektorientierten Programmierung vorgestellt und dies als Folge der immer größer werdenden Projekte dargestellt. Im Kern des Beitrags geht es jedoch um die richtige Programmiersprache für den Informatikunterricht. In einer Replik zu diesem Beitrag im folgenden Jahr [Sp99] wird daran jedoch kritisiert, dass nicht die Programmiersprache, sondern die Konzepte der objektorientierten Programmierung im Vordergrund stehen müssen. Als Beispiel dient das Konzept einer einführenden Stunde. Auch hier werden dann nur die Grundkonzepte vorgestellt. Die Wichtigkeit der objektorientierten Analyse (OOA) und des objektorientierten Designs (OOD) wird jedoch betont, ohne darauf weiter einzugehen.
Derselbe Autor hat im Jahr zuvor [Sp98] bereits das Modellieren in den Mittelpunkt gestellt und betont, dass die Architektur stimmen muss. Als Beispiel führt er das Modell-View-Control-Entwurfsmuster auf, welches als gute Architektur die Klassen der Verarbeitung (das Fachkonzept, Modell), die Ausgabe (View) und die Verwaltung der Eingaben (Control) trennt. Java wird als Unterrichtssprache empfohlen. In einer Replik darauf [vl99] wird der Verwendung von Java als Lernsprache widersprochen, der objektorientierte Softwareentwurf auf die Verwendung von Entwurfsmustern reduziert.

In den Jahren 2000 und 2001 ist deutlich zu erkennen wie wichtig das Thema geworden ist. Es sind mehr und etwas längere Beiträge zu finden.
In [Ba00] wird ein Unterrichtsentwurf zur Modellierung von Klassen vorgestellt. Wie sind Objekte, Klassen und Attribute zu finden? Auch das Suchen von Assoziationen wird vorgestellt, allerdings ist alles stark am Programmcode orientiert und es erfolgt keine Diskussion anhand eines abstrakteren Modells wie es z.B. das Klassendiagramm darstellt.
Ein Artikel nimmt dann den Konflikt *prozeduraler versus objektorientierter Entwurf* auf [Sp00]. Er stellt einen Unterricht vor, in dem allerdings der Lehrer den Schülern das entworfene Modell (ein Klassendiagramm) vorgibt und daran die objektorientierten Konzepte erklärt. Der Lernende setzt dann das Modell nur um.
Ein Vorschlag zur Modellierung ist dann in [Do00] die Umsetzung der Wiener Hofgesellschaft in einer Klassenhierarchie. Dabei wird von einem Text ausgehend das Problem analysiert, in ein Klassendiagramm umgesetzt und dann programmiert. Die einzelnen Objekte

werden dabei in einer Liste abgelegt, wobei dann Polymorphie zu Tage tritt. Beziehungen zwischen den Klassen kommen nicht vor.

T. Brinda schlägt in [Br00] bzw. [Br01] eine Klassifikation von Übungsbeispielen vor. Er möchte damit erreichen, dass eine Sammlung von Aufgaben für den Informatikunterricht entsteht. Für die statischen Modelle der Unified Modeling Language (v.a. die Klassendiagramme) entwirft er unterschiedliche Klassen, welche sich einerseits in der Fachlichkeit, v.a. aber in der Problemlösungstiefe unterscheiden. Er geht also von unterschiedlichen Lernzieltaxonomien aus und möchte mit weiter gehendem Unterricht immer tiefer gelangen. In allen seinen Beispielen spielt der Programmcode überhaupt keine Rolle. Zum Teil werden dadurch seine Beispiele recht komplex und abstrakt. Der Zusammenhang zu realen Programmen wird dem Lernenden dadurch kaum ersichtlich, die Inhalte sind v.a. bei den schwierigen Aufgaben stark theorielastig.
In [Sp01] wird auf das strukturierte Vorgehen im objektorientierten Softwareentwurf mit Analyse, Design, Implementierung und Test eingegangen. Im Rahmen eines Softwareprojekts sollen die Schüler dies umsetzen. Der Schwerpunkt des Artikels liegt jedoch auf der Organisation, Dokumentation und Bewertung der Projekte aus Lehrersicht und nicht auf der Modellierung.
Ein durchgängiges Konzept inkl. Assoziationen und Spezialisierungen liefert R. Baumann in [Ba01]. Es werden im ersten Schritt Klassen inkl. ihrer Attribute und Methoden gesucht, danach im Diagramm die Assoziationen untereinander eingetragen. Dabei spielt dann die Benutzeroberfläche (GUI) eine Rolle. Im nächsten Schritt wird nach Spezialisierungen gesucht, um so eine Vererbungshierarchie festzulegen. Dabei werden auch abstrakte Klassen eingeführt.
Im gesamten Unterrichtsentwurf sind sowohl Klassendiagramme als auch Java-Code zu finden. Die unterschiedlichen fachlichen Inhalte werden an unterschiedlichen Beispielen vorgestellt und an diesen auch umgesetzt.
Der Autor legt zwar auf die Modellierung Wert und zeigt auch teilweise das entstandene Klassendiagramm. Im Endeffekt macht er jedoch die Modellierung wieder an der Programmiersprache fest. In einer als Projekt vorgeschlagenen Vertiefung verzichtet er auf die Klassendiagramme und verschenkt so die programmiersprachen-unabhängige Formulierung der Modelle. Eine Diskussion der Ergebnisse wird dadurch erschwert.
Zur Diskussion zum Thema *Programmiersprache* trägt ein weiterer Artikel bei [We01]. Er schlägt Flash-Animationen zum objektorientierten Modellieren vor. Das Konzept geht sehr stark auf das Thema *Aggregation* ein, der Autor selbst sieht die Einsatzmöglichkeiten nur innerhalb einiger Grundkonzepte der Objektorientierung.

Im Jahr 2002 gab es in der Fachpublikation *Log In* keine bedeutende Veröffentlichung zum Thema. Dies ist verwunderlich, da ein durchgängiges modellierungsorientiertes Konzept immer noch fehlt.

3 Entwicklung des Unterrichtskonzepts

3.1 Didaktische und methodische Leitgedanken

Im neu zu erstellenden Unterrichtskonzept soll das Modellieren und die Problemlösekompetenz im Vordergrund stehen, wie sie in der einschlägigen informatischen Didaktik zwar oftmals gefordert, aber kaum zu finden sind (siehe Kapitel 2.2). Beides sind für unsere heutige Gesellschaft wichtige Fähigkeiten, sei es in den Ingenieurwissenschaften, den Naturwissenschaften, der Betriebswirtschaft oder in der Soziologie. Hier soll dies im Bereich des objektorientierten Softwareentwurfs vermittelt werden.
Die Codierung soll dabei aber nicht vergessen werden. Denn erst mit dem fertigen Programm ist das Problem gelöst und das Modell verifizierbar. Weiterhin ist das fertige Programm wichtig für die Motivation der Lernenden. Sie sehen, was sie erreicht haben, das schafft Selbstvertrauen und ein Selbstwertgefühl. Außerdem behält so die Modellierung Bodenhaftung und bei aller Abstraktheit einen realen Bezug.
Die 3 Kategorien, die meist in der Literatur getrennt vermittelt werden (siehe Kapitel 2.1), sollen gemischt werden. Dadurch wird das Konzept handlungsorientierter, da in der realen Softwareentwicklung alle Kategorien vorkommen. Diese Mischung soll so vorgenommen werden, dass sie jeweils dann vorkommen, wenn sie fachwissenschaftlich oder didaktisch benötigt werden, also immer in den entsprechenden Handlungsrahmen des Unterrichtsfortschritts hineinpassen.
Handlungsorientierung wird hier demnach im Sinne von *„orientiert sich nicht an der Fachsystematik, sondern an Handlungssituationen“* [Ba96, S. 183] verstanden. Dadurch sollen Zusammenhänge klarer verstanden und begründet werden.

Die in der modellierungsorientierten Literatur vorkommende Reihenfolge, dem realen Softwareentwurf folgend von den Anforderungen über die Analyse und das Design zur Programmierung, ist für den Unterricht aus didaktischen Gründen ungeschickt. Es können kaum die Anforderungen und noch weniger die Analyse oder das Design verstanden werden, wenn nicht klar ist, was überhaupt umsetzbar ist. Im realen Softwareentwurf steht immer ein Informatiker (also ein Wissender) als Fachmann zur Seite.
Innerhalb eines Buchs ist dieses Vorgehen noch tragbar, da der Leser sich nicht an die Reihenfolge der Kapitel halten muss und Teile des Buchs jederzeit wiederholt lesen kann. Im Unterricht ist dies aber weniger möglich, da der Lehrende die Reihenfolge über eine längere Zeit festlegt.
Die fachlich korrekten Strukturen ergeben also nicht automatisch auch die didaktischen Strukturen zur Vermittlung der Inhalte. Im Forschungsbereich der 'Didaktischen Rekonstruktion' wird diese Trennung bewusst vollzogen und dafür eine schülergerechte didaktische Struktur gefordert [Ka97]. Dazu vergleicht dieser Forschungsbereich die fachlichen Strukturen mit den Vorstellungen der Lernenden, mit dem Ziel, eine angepasste didaktische Struktur zu finden (z.B. in [Gr01]).

Diesem entsprechend wird eine im Gegensatz zur Literatur (siehe Kapitel 2) umgekehrte Reihenfolge zur Vermittlung vorgeschlagen. Der Lernende wird also über die Programmierung, das Design und die Analyse zu den Anforderungen geführt. Allerdings wird die Programmierung immer nur so weit geführt, um das gerade behandelte Design oder die Analyse zu verstehen, nicht weiter, damit Handlungsorientierung entsteht.

Damit kann der Lernende eher die neuen Inhalte an Bekanntem anknüpfen und alles besser festigen. Er wird dort abgeholt, wo er steht und wird langsam, wie über eine Treppe Stufe für Stufe zu Höherem geführt. Am Ende des Lernprozesses kann er hinabblicken und die Zusammenhänge besser verstehen.
Selbstverständlich sollen auch die Grundkonzepte des objektorientierten Softwareentwurfs vermittelt werden. Die Modellbildung soll primär an den graphischen Modellierungswerkzeugen der UML festgemacht werden, allerdings auch immer einen Bezug zur Codierung behalten. Die dynamische Modellierung per Sequenzdiagramm und Zustandsdiagramm soll ebenfalls Raum erhalten, da sie einerseits wichtige Prozesse im Softwareentwurf abbilden, andererseits aber auch exemplarisch für allgemeine, zeitabhängige Modelle stehen. Diese Art der Modellierung ist in der informatisch didaktischen Literatur kaum zu finden (siehe Kapitel 2.2).

Aus den oben aufgestellten Überlegungen ergeben sich dann die didaktischen und methodischen Leitgedanken, welche beim Entwurf des Konzepts im Vordergrund stehen:

1. Die Modellbildungskompetenz und Problemlösefähigkeit der Lernenden ist zu fördern.
2. Es soll ein vollständiges Konzept vorgelegt werden, welches den objektorientierten Softwareentwurf in allen Facetten – wenn auch nur exemplarisch, d.h. von der Ermittlung der Anforderungen über die Analyse und dem Design zur Codierung, also zum lauffähigen Programm – berücksichtigt. Die Lernenden sollen dabei dazu befähigt werden, ausgehend von einer kurzen und evtl. vagen Problemstellung wie z.B. *Ein Programm zum Memory-Spielen* selbständig und strukturiert die Lösung in Form eines lauffähigen Programms zu finden. Sie sollen lernen, die noch fehlenden Anforderungen selbständig zu beschaffen.
3. Dem Lernenden sollen nicht nur die Inhalte und Methoden vermittelt werden, sondern auch der Sinn und die Vorteile dieser Inhalte und Verfahren deutlich werden. In diesem Sinne ist das Ziel handlungsorientiert anzugehen. Neue Inhalte werden dann eingeführt, wenn es das Problem verlangt, bzw. die Probleme so gestellt, dass die Inhalte entsprechend eingeführt werden.
4. Die Reihenfolge der Stoffvermittlung soll dem Verlauf beim objektorientierten Softwareentwurf entgegenlaufen. Die Schritte: *Anforderungen, Analyse, Design* und *Programmierung* werden im Unterricht in der entgegengesetzten Reihenfolge durchlaufen. Dies hat den Vorteil, dass den Lernenden z.B. bei der Anforderungsanalyse bereits die restlichen Schritte bekannt sind und diese so besser verstanden werden. Die Lernenden sollen dort abgeholt werden, wo sie stehen und Stück für Stück in ihrer Erkenntnis und ihrer Methodik wachsen.
5. Auch wenn im Vordergrund die Modellbildung steht, sollen die modellorientierten und codierungsorientierten Anteile gemischt werden. Dieser Methodenwechsel soll das Konzept für den Lernenden interessanter machen und zur Stärkung der Motivation führen.
6. Das Konzept soll unabhängig von einer Programmiersprache zu realisieren sein. Das bedeutet nicht, dass nicht codiert werden soll, sondern dass eine beliebige objektorientierte Sprache verwendet werden kann. Dadurch sind auch die Lernziele allgemeiner.

3.2 Bezug zu den Anforderungen der Bildungseinrichtungen

Es soll nun untersucht werden, wie die oben genannten Leitgedanken des Konzepts zu den Anforderungen der Sekundarstufe II und zur universitären Ausbildung passen, da das neue Konzept für beide Einrichtungen verwendet werden soll.

Zuerst soll die Sekundarstufe II untersucht werden. Im Gymnasium wird u.a. eine

> *.. breite und vertiefte Allgemeinbildung gefordert, die zur Studierfähigkeit führt. Es fördert insbesondere die Fähigkeiten, theoretische Erkenntnisse nachzuvollziehen, schwierige Sachverhalte geistig zu durchdringen sowie vielschichtige Zusammenhänge zu durchschauen, zu ordnen und verständlich vortragen und darstellen zu können .*[6]

Was dabei unter *Allgemeinbildung* zu verstehen ist, bleibt etwas unklar. Ein Hinweis hierauf liefert die Expertise *Zur Entwicklung nationaler Bildungsstandards*, in welcher die grundlegende Idee der Allgemeinbildung als [Kl03, S. 59]

> *... die selbständige Teilhabe an Gesellschaft und Kultur ermöglichen, und ... zugleich einen Mindeststandard an kultureller Gemeinsamkeit, die „Basisfähigkeiten", sichern ...*

beschrieben wird. Da *breite* und *vertiefte* Allgemeinbildung gefordert wird, bedeutet eine selbständige Teilhabe an der Gesellschaft auch eine Teilnahme an der Weiterentwicklung dieser. Dazu müssen heutzutage viele Probleme in allen möglichen Disziplinen und Bereichen gelöst werden. Diese Problemlösefähigkeit kann jedoch nur fachspezifisch bzw. exemplarisch oder wie es obige Expertise nennt, innerhalb einer *Domäne*, vermittelt werden. Auch in der Pisa-Studie [Bm01], wird diese funktionale Auffassung der Allgemeinbildung vertreten und u.a. innerhalb der Domäne der *Mathematik* Kompetenzen der Schüler untersucht. Dazu wird der Begriff *Mathematical Literacy* verwendet, in dessen Kern das *Anwenden, verstanden als Prozess des Modellierens von Situationen mithilfe mathematischer Begriffe* [Bm01, S. 142], steht. Dies entspricht dem ersten Leitgedanken aus Kapitel 3.1, dass die Modellbildungskompetenz und Problemlösefähigkeit der Lernenden gefördert werden sollen. Allerdings hier nicht in der Domäne der Mathematik, sondern in der damit stark verwandten Domäne der Informatik.
Konkretisierungen der Kompetenzen, welche aus der *Mathematical Literacy* folgen, hat der *National Council of Teachers of Mathematics* (NCTM) aufgestellt [nach Bm01, S. 25]:

- *Vorbereitung auf offene Aufgabenstellungen, da realistische Probleme und Aufgaben in der Regel nicht gut definiert sind,*
- *Fähigkeit, die Anwendbarkeit mathematischer Konzepte und Modelle auf alltägliche und komplexe Problemstellungen zu erkennen,*
- *Fähigkeit, die einem Problem zu Grunde liegende mathematische Struktur zu sehen,*
- *Fähigkeit, Aufgabenstellungen in geeignete Operationen zu übersetzen, sowie*
- *ausreichende Kenntnisse und Beherrschung von Lösungsroutinen.*

6 Schulgesetz für Baden-Württemberg (SchG) §8, nach [Ho93].

Auch diese Konkretisierungen passen, wenn man die Domäne in die Informatik verlegt. So soll dieses Konzept besonders auf die Bearbeitung offener Fragen vorbereiten (siehe den zweiten Leitgedanken auf S. 13), wobei die Fragestellungen dabei so offen sind, dass die Schüler gezwungen sind, sich weitere Informationen selbständig, aber auf strukturierten Wegen zu besorgen. Des Weiteren sollen die Schüler mit Hilfe dieses Konzepts befähigt werden, informatische Konzepte und Modelle auf alltägliche und komplexe Problemstellungen anzuwenden. Um entsprechende Software zu entwerfen müssen sie dazu zunächst die Problemstellungen in informatische Modelle umformen. Dadurch erhalten sie auch die Fähigkeit, die entsprechenden informatischen Strukturen zu sehen und üben die Beherrschung der Lösungsroutinen. Aus obigem wird klar, dass damit auch die Konkretisierungen zur *Mathematical Literacy* des NCTM erfüllt werden.

Um zur fertigen Software, dem eigentlichen Ziel des Softwareentwurfs, zu gelangen, muss die Problemstellung zuerst modelliert werden (siehe Abbildung 3). Dazu wird zuerst ein Modell entworfen, welches das Problem abzubilden versucht, aber noch keine Lösung darstellt. Dieses Modell wird in der Softwaretechnik als Analysemodell bezeichnet. Im nächsten Schritt wird eine Lösung gesucht und diese in Form eines Designmodells abgebildet. Danach wird dieses Designmodell codiert, d.h. in einer Programmiersprache umgesetzt. Noch immer befinden wir uns dabei im Bereich der Informatik. Das auf einem Rechner lauffähige Programm liefert dann die Lösung des Problems. Nun kann man diese Lösung validieren, indem man sie mit dem Ausgangsproblem vergleicht und untersucht, ob das Problem korrekt gelöst wurde. Dieser Zyklus kommt dem *Prozess des Mathematisierens*, wie er in der Pisa-Studie vorgestellt wird, sehr nahe [Bm01, S. 144], weswegen er hier auch *Prozess des Softwareentwurfs* genannt wird. Interessant ist dabei die Möglichkeit des iterativen Vorgehens bis zur Lösung, da dieser Zyklus mehrfach durchlaufen werden kann. Dieses iterative Vorgehen wird in der Zwischenzeit beim Softwareentwurf, und damit auch in diesem Konzept, propagiert (siehe auch Abbildung 1 und Abbildung 2).

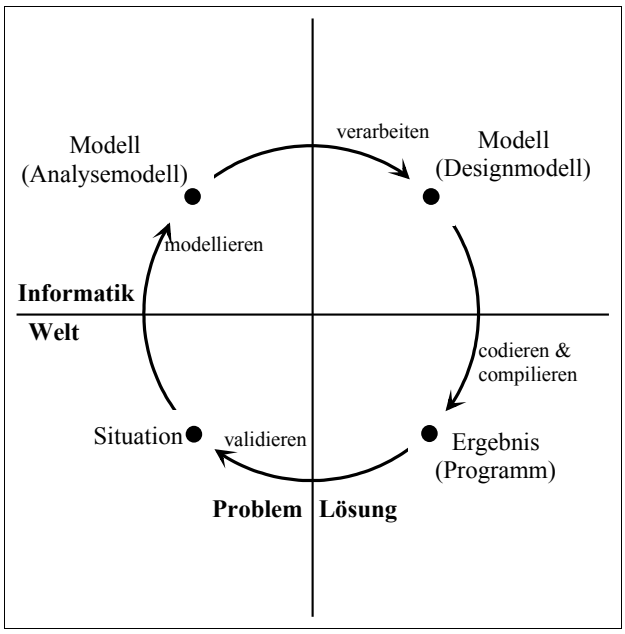


Abbildung 3. Der Prozess des Softwareentwurfs

Das hier vorgestellte Konzept zeigt hier also einen hohen Bildungswert im Sinne der Pisa-Studie und der Konkretisierung durch das *National Council of Teachers of Mathematics* (*NCTM*).

Das Schulgesetz (s.o.) konkretisiert jedoch auch selbst die Anforderungen, indem verlangt wird

1. *... theoretische Erkenntnisse nachzuvollziehen,*

2. *schwierige Sachverhalte geistig zu durchdringen sowie*
3. *vielschichtige Zusammenhänge zu durchschauen, zu ordnen und verständlich vortragen und darstellen zu können.*

Auch diese Forderungen werden vom hier vorgestellten Konzept erfüllt. Erstens werden theoretische Erkenntnisse aus dem Bereich der Informatik vermittelt, welche anschließend durch eigenständiges Modellieren der gestellten Probleme durch die Schüler nachvollzogen werden. Zweitens zeigen die Schüler dadurch, dass sie die recht abstrakten Inhalte geistig durchdrungen haben und drittens, dass sie damit die Zusammenhänge durchschauen.
Durch die Darstellung der Modelle und damit auch der Lösungen[7] mit Hilfe genormter Symbole (hier: **U**nified **M**odelling **L**anguage) können die Schüler diese kommunizieren und über Alternativen diskutieren. Diese Darstellung hilft auch die Zusammenhänge geordnet darzustellen.
Es werden also auch diese Forderungen erfüllt. Dies ist jedoch nicht verwunderlich, da, wie oben bereits begründet, das Konzept auch die Anforderungen an die Allgemeinbildung aus Sicht der Pisa-Studie erfüllt. Diese wiederum entspricht ebenfalls dem Schulgesetz bzw. geht z.T. darüber hinaus. Somit erfüllt das Konzept die Anforderungen der Pisa-Studie und des NCTM sowie des Schulgesetzes.

Interessant ist eine Untersuchung, ob das hier vorgestellte Konzept auch Klafkis Fragen bei der Auswahl allgemeinbildender Inhalte stand hält. Nach Klafki müssen dazu folgende Fragen beantwortet werden (nach [Hu01] S. 57):

1. *Lässt der (geplante) Inhalt zu, dass meine Schüler eine allgemeine Kenntnis, Einsicht erwerben können?*
2. *Ist der Inhalt so strukturiert, dass er neben seiner Besonderheit auch ein über sich hinausweisendes Merkmal aufweist?*
3. *Lässt sich das Allgemeine an diesem Inhalt auch von meinen Schülern in dieser Lernsituation erfassen?*
4. *Sollten meine Schüler dies Allgemeine überhaupt erwerben?*

Bevor diese Fragen in Bezug auf die hier vorgestellte Arbeit jedoch beantwortet werden, sollen darin einige Begriffe betrachtet werden[8]. Klassischer Weise wird in der Pädagogik unter *Inhalt* der Fachinhalt verstanden, hier also z.B. die in Kapitel 3.4 vorgestellten *Ziele und Inhalte der Objektorientierung*. Auch der Begriff *Kenntnis* wird als reines Wissen verstanden, wie es z.B. Blooms Lernzieltaxonomien (siehe Kapitel 3.5) als unterste Ebene vorsieht. Erweitert man jedoch den Begriff *Kenntnis* auch auf Prozesswissen, auf Fertigkeiten und Erfahrungen, also in einer Art und Weise wie er z.B. in der Psychologie[9] verwendet

7 Zum Teil genügt es, die Lösung nur im Bereich der Informatik und nicht in der realen Welt in Form von lauffähigen Programmen vorliegen zu haben. Dies ist aber nur möglich, wenn die Schüler bereits Erfahrungen mit der Umsetzung in lauffähige Programme gesammelt haben. Diese Erfahrungen strebt u.a. dieses Konzept an, indem immer wieder die von den Schülern gefundenen Lösungen codiert werden.

8 Diese Betrachtung entspricht evtl. nicht der eigentlichen Intention Klafkis, zeigt aber, dass diese Fragen immer noch aktuell sind.

9 Laut *Psychologischem Wörterbuch* [Do87] gibt es versch. Arten von Kenntnissen: Sach-Kenntnis, Verhaltens- oder Verfahrens-Kenntnis, Norm-Kenntnis, Wert-Kenntnis. Allgemein: Wissen, Fertigkeiten und Erfahrungen auf bestimmten Gebieten.

wird, so erscheinen Klafkis Fragen in einem etwas anderen Licht. Auch der Begriff *Inhalt* soll nun in diesem Sinne verstanden werden, also auch die Fertigkeiten und Erfahrungen einschließen. Es wird nun auch nach Kompetenzen gefragt, welche vermittelt werden sollen. Der Kompetenzbegriff soll dabei nach Weinert [Weinert, nach Kl03, S. 21] verwendet werden, also

> *die bei Individuen verfügbaren oder durch sie erlernbaren kognitiven Fähigkeiten und Fertigkeiten, um bestimmte Probleme zu lösen, sowie die damit verbundene motivationale, volitionale und soziale Bereitschaft und Fähigkeit, um die Problemlösungen in variablen Situationen erfolgreich und verantwortungsbewusst nutzen zu können.*

In diesem Sinne sollen nun Klafkis Fragen beantwortet werden. Seine erste Frage nach dem Erwerb einer allgemeinen Kenntnis muss bejaht werden, da hier die Modellbildungskompetenz und Problemlösefähigkeit gefördert werden soll. Beide sind für unsere heutige Gesellschaft wichtig und werden allgemein verlangt. Wie oben bereist erwähnt, ist heute eine wichtige Aufgabe die Weiterentwicklung der Gesellschaft. Dazu müssen jedoch die vielfältigsten Probleme in den unterschiedlichsten Fachbereichen gelöst werden. Um bei der oftmals vorkommenden Komplexität der Probleme zu einer Lösung zu kommen, sind häufig Modelle notwendig, in welchen auf die Problemdomäne abstrahiert wird. Es soll also eine allgemeine Kenntnis erworben werden, hier allerdings exemplarisch im Bereich der Objektorientierung.
Daraus stellt sich dann sofort Klafkis zweite Frage, ob die exemplarische Vermittlung an der Objektorientierung auch über sich hinausweist. Auch dies ist zu bejahen, da die Modellbildung und das Problemlösen systematisch angegangen werden sollen, d.h. es soll versucht werden, dass die Lösungen auf strukturierten und gut durchdachten Wegen gesucht werden. Dies weist über die Objektorientierung und die Informatik weit hinaus.
Die dritte Frage ist schwieriger zu beantworten, da hier eigentlich nur Vermutungen angestellt werden können. Im Rahmen der Evaluation des Konzepts soll sich zeigen, dass die Schüler die Inhalte im Bereich der Objektorientierung erfasst haben. Ob sie es allerdings im Allgemeinen, im Sinn der Frage eins, erfasst haben, ist jedoch sehr schwer nachzuweisen. Es ist jedoch anzunehmen, dass ein positives Resultat im Bereich der Objektorientierung auch ein positives Resultat im Allgemeinen mit sich bringt. Wie stark dieser Zusammenhang ist, läßt sich jedoch kaum beantworten.
Klafkis vierte Frage wurde bereits bei Frage eins behandelt, indem klar wurde, wie wichtig diese Inhalte für die Zukunft unserer Gesellschaft sind.
Als Ergebnis dieser kurzen Untersuchung zeigt sich also, dass auch im Sinne Klafkis hier Allgemeinbildung vermittelt wird, sofern wie oben beschrieben die Begriffe *Inhalt* und *Kenntnis* neu definiert werden. Diese Neudefinition ist jedoch in Bezug auf die Forderungen wie sie etwa die Pisa-Studie oder die Expertise *Zur Entwicklung nationaler Bildungsstandards* zeigen, sinnvoll.

In neueren Arbeiten erweitert Klafki sein Konzept und fordert eine Selbstbestimmungs-, Mitbestimmungs- und Solidaritätsfähigkeit von den Schülern. Er benennt diese genauer als *„epochaltypische Schlüsselprobleme unserer kulturellen, gesellschaftlichen, politischen, individuellen Existenz“* (nach [Ja02] *S. 231)*. Eines dieser Schlüsselprobleme sind nach Klafki heute „*die Gefahren und Möglichkeiten der neuen technischen Steuerungs-, Infor-*

mations- und Kommunikationsmedien" (nach [Ja02] S. 231). Auch in diesem Rahmen ist das Konzept allgemeinbildend, da die Schüler durch Beschäftigung mit der Objektorientierung und dadurch auch der Herstellung von Software implizit die Möglichkeiten der heutigen Informations- und Kommunikationsmedien besser einschätzen können.

Es zeigt sich also, dass das hier vorgestellte Konzept die Anforderungen an die Sekundarsufe II erfüllt. Dies einerseits aus Sicht der aktuellen Forschung im Rahmen der Pisa-Studie als auch aus Sicht von Klafki, sofern man die Begrifflichkeiten aktualisiert.

Als nächstes wird der Bezug des neuen Konzepts zu den Forderungen an eine universitäre Ausbildung untersucht. Im Universitätsgesetz des Landes Baden-Württemberg findet sich [UG00]:

> *Lehre und Studium sollen den Studierenden auf ein berufliches Tätigkeitsfeld vorbereiten und ihm die dafür erforderlichen fachlichen Kenntnisse, Fähigkeiten und Methoden dem jeweiligen Studiengang entsprechend so vermitteln, dass er zu wissenschaftlicher Arbeit und zu verantwortlichem Handeln ... befähigt wird.*

Die fachlichen Kenntnisse, Fähigkeiten und Methoden werden für den objektorientierten Softwareentwurf im neuen Konzept sicherlich vermittelt und stellen hier also kein Problem dar. Der objektorientierte Softwareentwurf ist, wie die vielfältige Literatur in diesem Bereich zeigt, in der Zwischenzeit ein wichtiges Fachgebiet nicht nur innerhalb der Informatik, sondern auch in Bereichen anderer Fachbereiche, welche einen Bezug zur Informatik haben. Hier sind u.a. die Ingenieurwissenschaften und die Betriebswirtschaft zu nennen. Interessanter ist die Untersuchung, ob wissenschaftliches Arbeiten vermittelt wird. Im hier vorgelegten Konzept liegt ein Schwerpunkt darauf, wie man zuerst informatische Probleme genauer analysiert und sie dann strukturiert in einem Top-Down-Ansatz durch Zerlegung in kleinere Einheiten überführt. Dieses Vorgehen wird auch oft bei wissenschaftlichem Arbeiten verlangt, um z.B. naturwissenschaftliche Phänomene oder mathematische Probleme zu klären. Insofern wird auch dieser Punkt der Anforderungen an eine universitäre Ausbildung erfüllt.

Dass die Leitgedanken sowohl den Forderungen des sekundären als auch des tertiären Bildungsbereichs entsprechen, soll nicht darüber hinwegtäuschen, dass sich die konkrete Umsetzung in den beiden Bereichen stark unterscheidet. Es soll aber zeigen, dass das grobe Konzept, wie es aus den Leitgedanken heraus entwickelt wird, unabhängig von der Bildungsstufe ist. Die Abhängigkeit von der Bildungsstufe zeigt sich erst in der konkreteren Umsetzung des Konzepts.

3.3 Voraussetzungen

Neben den Leitgedanken wie sie in Kapitel 3.1 entwickelt wurden, sind für die Entwicklung des Konzepts die Voraussetzungen der Lernenden zu klären, welche sie vor Beginn der Umsetzung des Konzepts mitbringen sollen.
Es wird davon ausgegangen, dass grundlegende Programmierkenntnisse bei den Lernenden vorhanden sind. Die Verwendung von Sequenzen, Auswahlen und Wiederholungen sollte in Theorie und Praxis bekannt und erste Erfahrungen mit Algorithmen erworben worden sein.

Im Rahmen der Umsetzung in der Klasse 12 des informationstechnischen Gymnasiums in Baden-Württemberg müsste dies durch den Unterricht in Klasse 11 gewährleistet sein [It01][Ai01]. Dies soll jedoch auch die Evaluation zeigen.
Solange innerhalb der Methoden die Algorithmen einfach gehalten werden, sind die oben genannten Anforderungen nicht sehr streng. Es ist sogar zu überlegen, auf die genannten Voraussetzungen in Gänze zu verzichten und Auswahl und Wiederholung während der Einführung in den objektorientierten Softwareentwurf mit einfließen zu lassen bzw. in einer anschließenden Einheit zu vermitteln. In diesem Fall müsste das Unterrichtskonzept angepasst werden. Es ist jedoch anzunehmen, dass dies keinen grundsätzlichen Neuentwurf des Konzepts benötigt, sondern nur Einschübe mit den entsprechenden Inhalten verlangt. Dies könnte man wiederum − entsprechend dem Grundgedanken der Handlungsorientierung − an Beispielen, bei denen diese benötigt werden, einbauen. So könnte das strukturierte Programmierkonzept der Wiederholung beim sequentiellen Einlesen einer Datei eingeführt werden.
Da das Unterrichtskonzept jedoch in einer Umgebung umgesetzt werden soll, bei dem die obigen Voraussetzungen bestehen, wird bei der Konzepterstellung davon ausgegangen, dass sie vorhanden sind und bei der Evaluation das Vorhandensein dieser überprüft.

Weiterhin sollten natürlich auch weitere Voraussetzungen, die nicht fachlicher Natur sind, einfließen. Dazu gehört vor allem das Lebensumfeld der Lernenden. Primär bedeutet dies, das die gewählten Beispiele passend ausgesucht werden sollen, um Verständnis und Motivation zu erreichen. Hier unterscheidet sich wiederum die Sekundarstufe II vom tertiären Bereich. Beispiele werden jedoch erst in der detaillierten Umsetzung gesucht, so dass das grobe Konzept für beide verwendet werden kann.
Weitere Voraussetzungen wie Gruppengröße und Zusammensetzung oder Arbeitsmittel (hier v.a. Rechnerausstattung) lassen sich nur schwer bei der Grobplanung berücksichtigen, da zu diesem Zeitpunkt diese meist nicht bekannt sind. Deswegen können diese Bedingungen meist erst in die Feinplanung eingehen. Allgemein ist zu sagen, dass es natürlich besser ist kleinere Gruppen zu haben, wobei jedem ein Rechner zur Verfügung stehen sollte. Dies ist aber oftmals nicht zu erreichen.

3.4 Ziele und Inhalte in der Objektorientierung

Bevor ein Konzept für den objektorientierten Softwareentwurf erstellt werden kann, müssen zuerst die Ziele und Inhalte festgelegt werden. Als Quellen wurde die in Kapitel 2.1 vorgestellte fachwissenschafliche Literatur verwendet.
Die einzelnen Inhalte stehen dabei in einer thematischen Beziehung untereinander. Dieses Beziehungsgeflecht lässt sich am besten in einem Baumdiagramm aufzeichnen, um so eine begriffliche Hierarchie und damit eine Struktur zu erhalten. Dabei werden die Abhängigkeiten unter den Begrifflichkeiten klarer. Da eine klare Einteilung zum Teil jedoch schwer fällt, wurden auch Verbindungen zwischen den Blättern des Baumes gelegt. Dies ist z.B. für den Begriff der Sichtbarkeit der Fall, welcher sich sowohl auf Attribute als auch auf Methoden bezieht.
Dieses Sammeln entspricht einer Analyse des Problems. Dabei soll zunächst kein Bezug auf die Zielgruppe erfolgen, um das Konzept vorerst unabhängig davon entwickeln zu können. Weiterhin ist anzumerken, dass die angegebenen Strukturen fachlicher Natur sind und kein Bezug zur didaktischen Reihenfolge bei der Vermittlung der Inhalte besteht, wie bereits in

Kapitel 3.1 erläutert wurde. Die dargestellte Struktur stellt also eine rein fachliche Gliederung dar.
Ein Problem bei der Inhaltssuche ergibt sich daraus, dass manche Inhalte stark programmiersprachenabhängig sind. Als Beispiel sei das Konzept der Friend-Klassen in C++ erwähnt. Dabei wird einer anderen Klasse der Zugriff auch auf private Attribute und Methoden gewährt. Dieses Konzept kommt in anderen Programmiersprachen wie Java oder Smalltalk nicht vor. Im Vordergrund des Unterrichtskonzepts sollen jedoch die Grundprinzipien und Verfahren der Objektorientierung stehen. Gerade deshalb soll das Unterrichtskonzept unabhängig von Programmiersprachen entworfen werden. Eine Aufnahme z.B. des obigen Inhalts (Friend-Klassen) würde diesem widersprechen, weswegen allgemein diese Art von Konzepten nicht aufgenommen wurde.
Da in der konkreten Umsetzung aus Motivationsgründen und zur Vertiefung eine Programmiersprache eingesetzt werden sollte, können diese Inhalte jedoch an den entsprechenden Stellen im Konzept eingebaut werden.
Ebenso wurden Konzepte, welche auch in der strukturierten Programmierung vorkommen, nicht aufgenommen, auch wenn sie evtl. für die objektorientierte Programmierung geringfügig angepasst wurden. Hierzu zählt zum Beispiel der Typecast, welcher bereits in C vorhanden ist, in C++ jedoch konsequenterweise auf den Cast von einer Unter- auf die Oberklasse erweitert wurde. Prinzipiell stellt dies jedoch kein besonderes, objektorientiertes Konzept dar, sondern ist eher das Merkmal einer typsicheren Sprache, wie es eben C++ und Java sind.

Die Inhalte lassen sich grob in zwei Bereiche einteilen: Die Konzepte der Objektorientierung und die Entwurfsmethoden (siehe Abbildung 4).

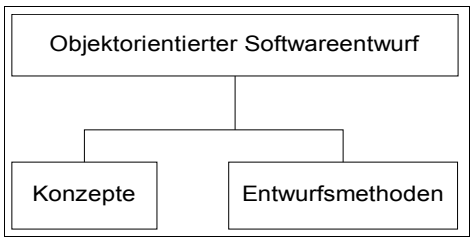


Abbildung 4. Die fachliche Einteilung des objektorientierten Softwareentwurfs

Im Bereich der Konzepte (siehe Abbildung 5) finden sich die Eigenschaften objektorientierter Sprachen. Dabei wird im obigen Sinne jedoch vermieden, spezielle Eigenschaften einer Sprache zu bevorzugen. Trotzdem stehen C++ und Java als sehr verbreitet vorkommende Vertreter im Vordergrund.
In Abbildung 5 finden sich die Konzepte der Objektorientierung im Detail. Dort sind bereits die Begriffe markiert, welche im Unterrichtskonzept vermittelt werden sollen und als Referenz, welche im Lehrplan des informationstechnischen Gymnasiums vorkommen. Eine Begründung für die Auswahl erfolgt unten.
Es ist anzumerken, dass das angegebene Konzept der *virtuellen Methode* in Java nur indirekt vorkommt, da dort alle Methoden virtuell sind.

Der Bereich der Entwurfsmethoden (siehe Abbildung 6) ist immer noch stark im Fluss, weshalb hier eine Sortierung schwer fällt. Deshalb ist die Struktur nicht eindeutig und die

Begriffe (vor allem die *Methoden*) stellen nur eine Auswahl dar. Das Baumdiagramm kann jedoch einfach erweitert und angepasst werden.

Da nur eine begrenzte Zeit zur Vermittlung der Inhalte zur Verfügung steht, wurden jene Konzepte gewählt, welche für die Verbesserung des problemlösenden Denkens der Lernenden wichtig und für die Entwurfsmethoden nötig sind. Weiterhin müssen natürlich jene Inhalte vermittelt werden, welche zur Umsetzung der objektorientierten Modelle in einfache lauffähige Programme nötig sind.

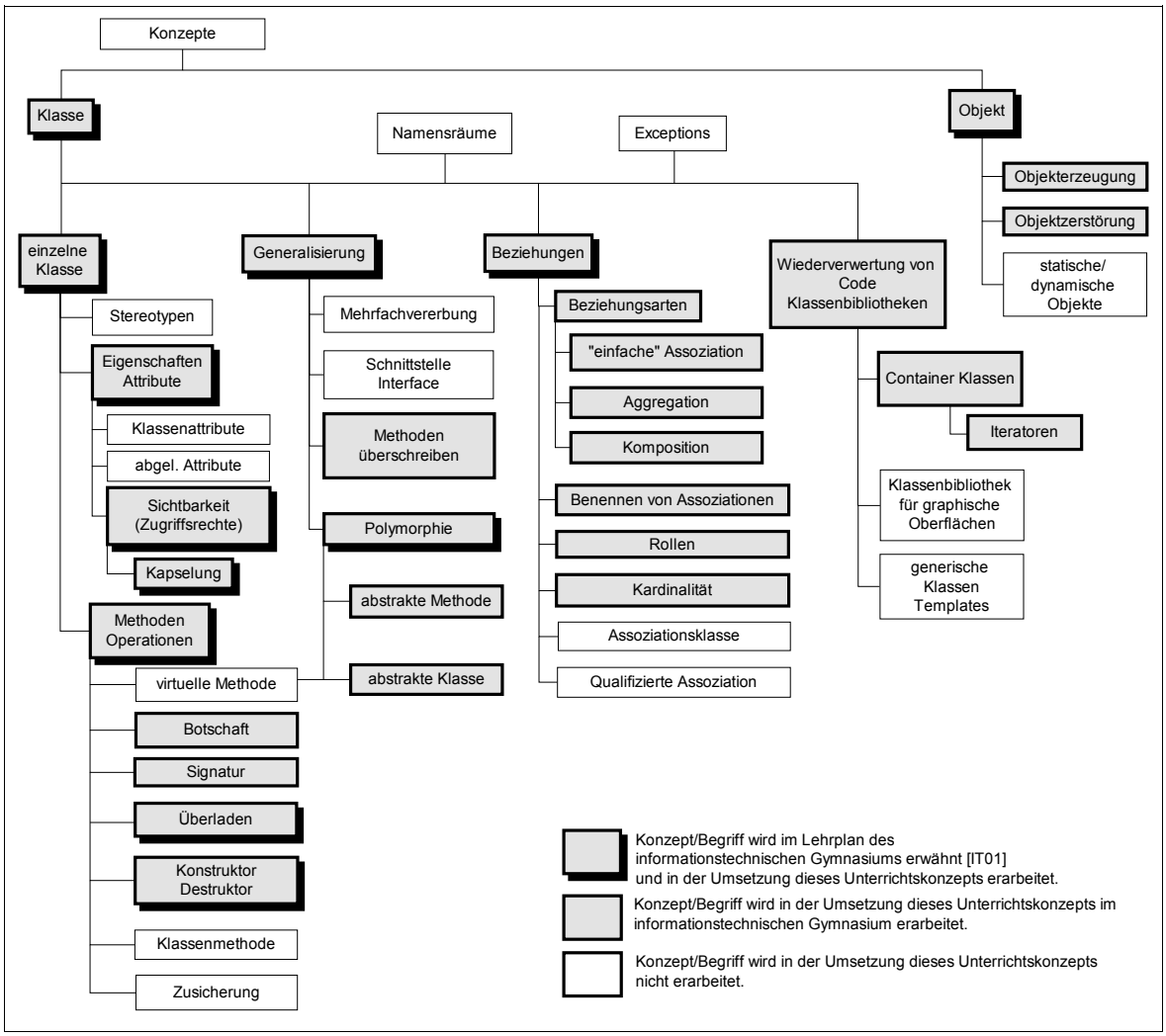


Abbildung 5. Die Konzepte der Objektorientierung

Im Gegensatz hierzu wurden Konzepte nicht aufgenommen, wenn sie für das problemlösende Denken weniger wichtig sind, auch wenn sie eine gewisse Bedeutung für die Programmierung haben. Zu nennen sind hier z.B. die *Exceptions*, welche zwar zum Abfangen von Fehlern wichtig sind und aus der objektorientierten Struktur folgen, aber bei den Modellen nicht im Vordergrund stehen.
Dies gilt auch für einige speziellere Konzepte, welche zwar in manchen Modellen hilfreich sind, aber nicht mehr in den zeitlichen Rahmen passen. Zu nennen wären hier die Klassenattribute und -methoden, die abgeleiteten Attribute, die Assoziationsklassen, qualifizierte

Assoziationen, Stereotypen und generischen Klassen. Manche Konzepte werden auch nicht von allen Programmiersprachen verwendet, wie z.B. die Mehrfachvererbung nicht bei Java, die Schnittstellen nicht bei C++.
Erst bei recht großen Projekten ist das Konzept der *Namensräume* sinnvoll. Für das grundlegende Verständnis des objektorientierten Softwareentwurfs ist es nicht nötig. Deswegen wird dieses Konzept hier nicht vermittelt.

Im Mittelpunk des objektorientierten Softwareentwurfs stehen natürlich die Begriffe *Klasse* und *Objekt*. Daraus ergibt sich dann auch, dass der innere Klassenaufbau (Attribute, Methoden) und die Kapselung der Daten (Information Hiding) als Grundkonzepte wichtig sind. Des Weiteren aber auch wie einzelne Klassen untereinander, entweder als Generalisierung oder als Beziehung in den verschiedenen Ausprägungen wie einfache Assoziation, Aggregation oder Komposition in Verbindung stehen. Für den eigentlichen Entwurf ist die Unterscheidung zwischen Aggregation und Komposition meist weniger wichtig, allerdings wird dadurch das Denken geschult, da sehr genau überprüft werden muss, ob die Beziehung existentieller Natur ist.
Für die Umsetzung in einer Programmiersprache ergibt sich aus diesen Grundkonzepten dann die Notwendigkeit, Objekte zu erzeugen und zu zerstören, und daraus das Konzept des Konstruktors bzw. Destruktors um Werte im Objekt zu initialisieren bzw. Aktionen bei der Objekterstellung auszuführen und vor Zerstörung des Objekts aufzuräumen.

Für den Entwurf ist es wichtig, diese Beziehungen genauer zu untersuchen, indem die Assoziationen Namen erhalten, um sie genauer zu beschreiben und um Rollen bzw. Kardinalitäten festzulegen. Mit Hilfe von Rollen kann z.B. eine Klasse mehrere Funktionen wahrnehmen: ein Mann kann einmal die Rolle eines Vaters, einmal die Rolle eines Sohns einnehmen. Das Konzept der Rolle hilft also, das Modell der Realität oder besser dem menschlichen Denken näher zu bringen. Mit den Kardinalitäten[10] kann genau festgelegt werden, ob z.B. Autos mit 3, 4 oder mehr Rädern berücksichtigt werden sollen oder nur solche mit 4. So lässt sich mit Rollen und Kardinalitäten ein Modell originalgetreuer erstellen und genauer spezifizieren.
Aus dieser Entscheidung folgen dann unterschiedliche programmiertechnische Umsetzungen. Feststehende Kardinalitäten werden am besten über einzelne Attribute bzw. Felder realisiert. Die Kardinalität *viele* wird am besten mit einer entsprechenden Container-Klasse[11] (zusammen mit den entsprechenden Iteratoren), welche bereits in den Softwarebibliotheken vorhanden sind, umgesetzt. So wird weiterhin der Gebrauch von vorgefertigten Klassen und die Wiederverwertung von Code einleuchtend.
Der Gebrauch vorgefertigter Klassen wird auch im Rahmen der Programmierung der graphischen Oberfläche geübt, allerdings erfolgt dies nicht explizit, sondern die Lernenden sollen angeregt werden, die Hilfe der Softwarebibliothek zu verwenden.

Eine völlig neue Möglichkeit bei der objektorientierten Programmierung ist die Polymorphie. Durch die dabei benutzte dynamische Bindung ist es möglich, dass erst zur Laufzeit bekannt ist, welche Methode bei einem Aufruf verwendet wird. Dadurch ist es möglich,

10 Begriff: siehe Glosar (Anhang E)

11 Containerklassen z.B. in Java in der Klasse java.util.Vector, in C++ in der Klasse vector der Standard Template Library [ISO98], bzw. bei älteren Entwicklungsumgebungen in entsprechenden Bibliotheken.

unterschiedliche Objekte in einem Container abzulegen (siehe oben) und entsprechend dem Objekttyp ohne die Verwendung von Auswahlen zu verarbeiten. So lassen sich Methoden verwenden, deren Existenz beim Entwurf des Aufrufers noch nicht bekannt waren. Es muss nur die Signatur, d.h. der Methodenname und die Aufrufparameter gegeben sein. Eine Folge sind abstrakte Methoden, welche keine Implementierung besitzen und in der Kindklasse überschrieben werden, damit sie dort erst verwendbar sind und abstrakte Klassen, von denen keine Objekte erzeugt werden können, da sie abstrakte Methoden besitzen. In C++ müssen diese Methoden des Weiteren rein virtuell sein, in Java sind bereits alle Methoden virtuell.

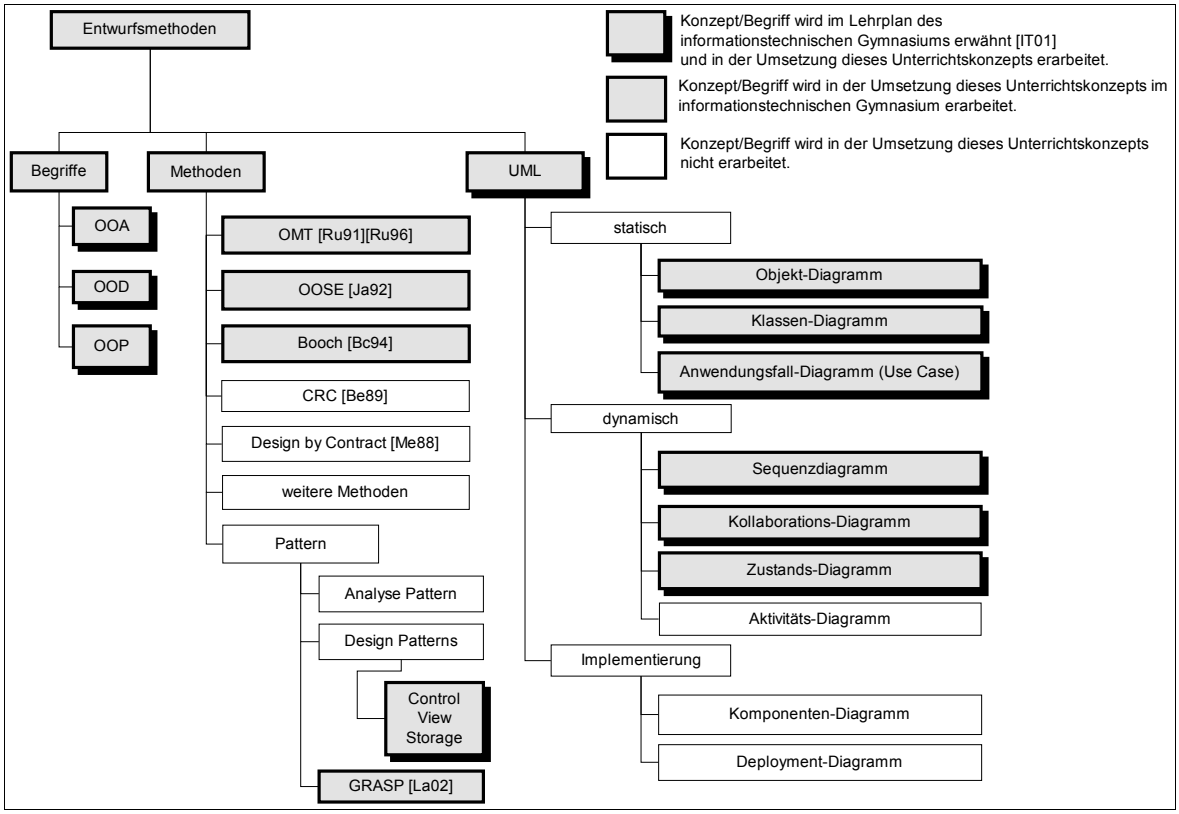


Abbildung 6. Die Entwurfsmethoden des objektorientierten Softwareentwurfs

Aus der Verbesserung der Problemlösefähigkeit folgt direkt das strukturierte und planvolle Arbeiten. Dabei ergibt sich automatisch das Dreigespann *Analyse*, *Design* und *Programmierung*, welches im Softwarebereich häufig verwendet wird (siehe Abbildung 6).
Um diese Schritte gut zu dokumentieren empfiehlt sich eine graphische Darstellung, wie sie sich in den UML-Diagrammen in einer genormten Art und Weise finden [OMG00]. Dabei ist das statische Klassendiagramm sicher das Wichtigste. An zweiter Stelle folgt das Sequenzdiagramm, welches dynamische Abläufe abbildet und über Botschaften zwischen den Objekten hervorragend zum Finden von Methoden geeignet ist. Weiterhin das Zustandsdiagramm, welches schon lange vor der objektorientierten Programmierung bekannt war und das Use-Case-Diagramm. Letzteres recht simple Diagramm besitzt seine Stärke in der übersichtlichen Darstellung der Anforderungen an die Software. Wichtiger als das Diagramm selbst ist jedoch die Vorgehensweise beim Finden der Anforderungen und die

Kommunikation des Softwareingenieurs mit dem Softwareanwender während der Aufstellung der Anforderungen.
Das Objektdiagramm ist nur zu Beginn wichtig, um die Unterscheidung zwischen Objekt und Klasse auch graphisch darstellen zu können. Entsprechend verhält es sich mit dem Kollaborationsdiagramm, welches dieselben Informationen wie ein Sequenzdiagramm enthält. Im Vordergrund steht dabei allerdings weniger der zeitliche Ablauf, sondern die Zusammenarbeit (Kollaboration) zwischen den Objekten.
Das Aktivitätsdiagramm wird vor allem zur Abbildung von Geschäftsprozessen verwendet und dient zur genaueren Beschreibung der Anwendungsfälle (Use Cases). Sinnvoll ist dessen Verwendung jedoch erst bei wesentlich größeren Projekten, weswegen es hier nicht vermittelt wird. Entsprechendes gilt für die beiden Implementierungsdiagramme.

Die Auswahl der Entwurfsmethoden fällt schwer. Es wurden hier die Verfahren ausgewählt, welche dann zur Unified Modeling Language führten: Dies ist einerseits Rumbaugh [Ru91][Ru96], welcher mit der Object Modeling Technique (OMT) neben der statischen Struktur auch die dynamische verwendete und versuchte, dies graphisch darzustellen, Jacobsen [Ja92], der den Use-Case-Driven Entwurf einführte – also den Entwurf basierend auf Anwendungsfällen – und Booch [Bc94], der an der strukturierten Suche der Klassen und ihrer Beziehungen und an der graphischen Darstellung jener arbeitete. Hinzu kommt die Methode von Abbott [Ab83], in welcher er eine Textanalyse zum Finden von Klassen vorschlägt. So sind Substantive Kandidaten für Klassen und Verben Kandidaten für Methoden. Hier könnten jedoch auch andere Verfahren gewählt werden, solange sie das strukturierte und planvolle Arbeiten der Lernenden unterstützen und fördern, um vom Problem zur Lösung in Form eines objektorientierten Programms zu finden.
Sehr hilfreich für den guten Entwurf ist die Kenntnis von Entwurfsmustern (Patterns), d.h. von ausgearbeiteten Klassenstrukturen zum Lösen häufig vorkommender Probleme. In diesem Sinne entsprechen sie den Standardalgorithmen der Informatik, wie z.B. den Sortieralgorithmen, da auch diese eine fertige, gut durchdachte Lösung für das Problem (hier das Sortieren) darstellen.
Allerdings sind Entwurfsmuster sehr vielfältig (siehe z.B. [Ga96]) und erfordern für ihre Erklärung auch die entsprechenden Problemstellungen. Deswegen sollten sie erst in der Vertiefung vorkommen. Des Weiteren sollte bereits etwas Erfahrung vorhanden sein, um die Patterns nachvollziehen zu können.
Anders verhält es sich mit den *General Responsibility Assignment Software Patterns* von C. Larman (GRASP, [La02]). Sie sind zumeist eher Regeln, um ein gutes Design zu erhalten. Von den insgesamt 9 GRASP Entwurfsmustern eignen sich dazu vor allem folgende:

Informations-Experte:	Welche Klasse ist für welche Informationen zuständig?
Geringe Kopplung:	Es sollten möglichst wenig Verbindungen zu anderen Klassen bestehen, um so bei Änderungen möglichst wenig Nachbarn zu beeinflussen.
Hoher Zusammenhalt:	Die Verantwortlichkeit einer Klasse sollte sich möglichst nur auf eine logische Aufgabe beziehen.
Erzeuger:	Die Klasse ist für die Erzeugung neuer Objekte zuständig, welche die zu erzeugenden Objekte beinhaltet, die Informati-

onen zum Start des neuen Objekts enthält oder das zu erzeugende Objekt eng verwendet.

Sie eignen sich auch, um unterschiedliche Entwürfe zu vergleichen und zu bewerten.

Bei der genannten Stoffauswahl ist zu berücksichtigen, dass nicht in die allerletzte Tiefe vorgedrungen wurde. Deswegen eignet sich diese Auswahl für den Gebrauch in der Sekundarstufe II mit ihrem allgemeinbildenden Anspruch oder für eine erste, aber durchaus solide Einführung in das Thema an einer Universität.
Für tiefer gehende Ansprüche muss im Anschluss noch weitergearbeitet werden. Für die Konzepte sind dann noch die generischen Klassen und die Exceptions hinzuzunehmen, für größere Projekte die Namensräume. Für die Methodik sollten noch weitere Verfahren, wie z.B. *Design by Contract* oder *Extreme Programming* eingeführt und selbstverständlich wichtige Designmuster wie z.B. *Factory*, *Facade* oder *Singleton* vermittelt werden [Ga96].

3.5 Lernzieltaxonomien

Neben den reinen Inhalten ist auch die Lernzieltaxonomie wichtig, d.h. ob das Lernziel nur reproduziert werden muß, ob von den Lernenden ein Wissenstransfer gefordert wird oder gar das Gelernte kreativ verwendet werden soll. Dies ist von Inhalt zu Inhalt unterschiedlich und ist davon abhängig, ob die Inhalte an einer Hochschule oder einem Gymnasium vermittelt werden sollen.
Es stellt sich nun also die Frage, wie tief welche Inhalte vermittelt werden sollen. Ein geeignetes Instrumentarium als Maß für diese *Lerntiefe* sind die Lernzieltaxonomien nach Bloom [Bl56]. Bloom stuft dabei die Verhaltensweisen, die der Lernende am Ende der Lernerfahrung mitbringen soll, ab. Er gliedert sie in 6 Stufen mit steigender Komplexität der Denkvorgänge[12]:

Wissen (knowledge):	Wissen bezieht sich hier auf das Abrufen von bekannten Inhalten. Es ist zu vergleichen mit dem Abrufen von Daten aus einer Datenbank. Die geistige Leistung besteht darin, den richtigen Schlüssel zu finden, um an die entsprechende Information zu gelangen.
Verstehen (comprehension):	Niedrigste Ebene des Begreifens. Das Individuum weiß Bescheid, über welche Dinge kommuniziert wird und kann damit arbeiten, jedoch ohne es mit Anderem in Verbindung zu bringen und seine umfassende Bedeutung zu erkennen.
Anwendung (application):	Die Anwendung von Abstraktionen auf konkrete Situationen, z.B. wie sich die Veränderung eines Parameters einer mathematischen Formel auf das Endergebnis auswirkt.
Analyse (analysis):	Das Zergliedern von Sachverhalten in ihre grundlegenden Elemente und die Strukturierung dieser Zergliederung mit dem Ziel, den Sachverhalt besser zu verstehen und zu klären.
Synthese	Das Zusammenfügen von Elementen zu einem Ganzen, so dass

12 Einige Taxonomiestufen werden von Bloom noch weiter untergliedert. Dies macht jedoch eine Verwendung dieser in der Lehrzielmatrix (s.u.) unhandlich, da zu viele Spalten entstünden.

(synthesis):	etwas entsteht, was vorher nicht klar erkenntlich war.
Evaluation (evaluation):	Beurteilen von Sachverhalten nach ihrem Wert zur Erfüllung bestimmter Zwecke. Die Kriterien können selbst bestimmt oder gegeben sein.

Eine Alternative zu den Lernzieltaxonomien von Bloom stellen die 4 Lernzielstufen der Empfehlung des Deutschen Bildungsrates dar [De72, S. 78]:

Reproduktion:	Das Gelernte auf Abruf durch Stichworte aus dem Gedächtnis wiedergeben. Eingeschlossen ist hierbei zum Teil das Verstehen, welches bei Bloom eine eigene Stufe darstellt.
Reorganisation:	Der Stoff soll selbst verarbeitet und neu strukturiert werden.
Transfer:	Grundprinzipien des Gelernten sollen auf neue ähnliche Aufgaben übertragen werden.
Problemlösendes Denken:	Es wird etwas aus Sicht des Lernenden völlig Neues geschaffen.

Die 6 Taxonomiestufen nach Bloom ermöglichen im Gegensatz zu den 4 Stufen des Deutschen Bildungsrats eine etwas feinere Aufgliederung der Tiefe der Vermittlung. Deswegen wird in dieser Arbeit mit den Taxonomien nach Bloom gearbeitet.

		Verhalten (Lernzieltaxonomien nach Bloom [Bl56])					
		Wissen	Verstehen	Anwendung	Analyse	Synthese	Evaluation
I	Konzepte						
n	Klasse & Objekt	A	B_2	C_1			
h	Beziehungen	D	E_2	F_4	G_1	H_5	I
a	Generalisierung	K	L_2	M_4	N_1	O_5	P
l	Methoden						
t	Klassenentwurf	Q	R	S_5			
e	Use Cases	T	U	W_5			
	UML	X		Y_3			

Tabelle 1. Lehrzielmatrix nach Taylor [Ty71] für das Unterrichtskonzept[13]

Die Inhalte und die gewünschten Verhaltensweisen lassen sich nun in die so genannte Taylorsche Lehrzielmatrix [Ty71] zusammenfügen, so dass zu jedem Inhaltsziel ein Verhaltensziel angegeben werden kann (siehe Tabelle 1).

Die im Kapitel 3.4 gefundenen Inhalte wurden aus Gründen der Übersichtlichkeit in die Konzepte *Klasse & Objekt*, *Beziehungen* und *Generalisierung* und die Methoden *Klassenentwurf* und *Use Cases* zusammengefasst. Der Inhalt *UML* wurde extra aufgeführt, da er einerseits mit den Methoden eng verknüpft ist, andererseits eine normierte Darstellungsform ist. Horizontal werden die Lernzieltaxonomien eingetragen.
Beim Ausfüllen der Tabelle wird dann festgelegt, welches Ziel mit welcher Taxonomie erreicht werden soll. Dies ist abhängig davon, für welche Zielgruppe geplant wird. Die

13 Buchstaben dienen zur späteren Referenzierung. Die markierten Felder entsprechen den hier gewünschten Zielen. Die Indizes geben die Lernziele des Lehrplans an (siehe Tabelle 2).

Inhalte für Studenten eines Studiengangs der Informatik und für Schüler eines informationstechnischen Gymnasiums können dabei identisch sein – auch wenn wahrscheinlich an einer Universität weitere Inhalte hinzukommen. Der Unterschied liegt dann aber in der Tiefe des Verständnisses oder eben in der Lernzieltaxonomiestufe, welche erreicht werden soll.

5	**Objektorientierte Analyse und Design**		**45**
5.1	Elemente der realen Welt auf Objekte als Modelle abbilden$_1$ und Basiskonzepte von Objekten beschreiben$_2$	Objekt- und Klassenstruktur - Attribute - Operationen Prinzip der Vererbung	Kapselung Synonym: Felder, Eigenschaften Synonym: Methoden Hierarchiebaum
5.2	Graphische Notation von Objektstrukturen$_3$ und Wechselwirkungen zwischen den Objekten anwenden$_4$	Anwendungsfalldiagramm Klassendiagramm Objektdiagramm Kollaborationsdiagramm Sequenzdiagramm Zustandsdiagramm	UML LPE 5.2 sollte zusammen mit LPE 5.3 vermittelt werden.
5.3	Lösungen für Probleme durch objektorientierte Analyse und objektorientiertes Design entwickeln und implementieren$_5$	Klassen Objekte Vererbung Sichtbarkeitskennzeichnung Konstruktoren, Destruktoren Überladen von Operationen Polymorphie 3-Schichten-Architektur	CASE-Tools public, private, protected Fachkonzept Benutzeroberfläche Datenhaltung

Tabelle 2. Lehrplaneinheit 5 des Fachs "Informationstechnik" des informationstechnischen Gymnasiums [It01][14]

Die angegebene Tabelle gilt für die Verwendung in einem informationstechnischen Gymnasium. Bei der Auswahl der Lernzieltaxonomien der einzelnen Inhalte stand die Modellbildung und das Problemlösen im Vordergrund wie dies im Kapitel 3.2 für das Gymnasium begründet wurde. So wurde für die Inhalte *Beziehungen* und *Generalisierung* die höchste Lernzieltaxonomie *Evaluation* gewählt, da es für die Erstellung eines guten Modells wichtig ist, dies mit anderen Modellen zu vergleichen und dabei die Vor- und Nachteile zu sehen. Dies führt dann in späteren Modellbildungsprozessen bei den Lernenden zu besseren Modellen.

Für den Inhalt des Klassenentwurfs wurde als höchste Stufe *Anwendung* gewählt. Die Taxonomiestufe der Synthese ist nicht angegeben, obwohl bei erster Betrachtung ein Klassenentwurf eine Synthese darstellt. Dies wird jedoch schon durch die Stufe *Anwenden* angegeben.

14 Die Indizes beziehen sich auf die Indizes in Tabelle 1.

Analyse und *Synthese* bezieht sich hierbei auf die Analyse des Klassenentwurfverfahrens. Man befindet sich hier also bereits auf einer Metastufe: die Analyse des Syntheseverfahrens. Dies führt für ein Gymnasium zu weit, wogegen hier das wissenschaftliche Arbeiten an einer Universität beginnt: Ein allgemein gültiges Verfahren zur Lösung allgemeiner (Software-) Probleme ist zu suchen und mit anderen Verfahren zu vergleichen (Evaluation).
Entsprechendes gilt für die *Use Cases*, mit deren Hilfe die Anforderungen an die spezielle Softwarelösung gefunden werden sollen.
Für den Inhalt *Klasse & Objekt* genügt die Stufe *Anwenden*, da einzelne Klassen oder Objekte ohne Bezug zu anderen Klassen bzw. Objekten nur in der Einführung Sinn machen. Später stehen diese immer über Beziehungen oder Generalisierungen in Bezug zueinander. Dort sind dann die höheren Lernzieltaxonomien gesetzt.
Der Inhalt *UML* soll auch nur angewandt werden, da keine neue Notation im Gymnasium gefunden werden soll (Synthese) oder diese graphische Notation mit ihren vielen Möglichkeiten analysiert werden soll. Die UML wird nur als genormtes Hilfsmittel gebraucht, um die gefundenen Modelle zu beschreiben und zu vergleichen.

Die Inhalte des Lehrplans des informationstechnischen Gymnasiums (siehe Tabelle 2) wurden ebenfalls in die Lehrzielmatrix übersetzt. Sind höhere Taxonomien angegeben, so ist zu beachten, dass auch die niedrigeren Stufen verlangt sind. Eine Synthese ist nur möglich, wenn der entsprechende Inhalt verstanden ist und angewandt werden kann.
Das hier vorgestellte Unterrichtskonzept geht etwas über den Lehrplan hinaus. Die Konzepte *Beziehungen* und *Generalisierung* sollen hier von den Lernenden evaluiert werden, hingegen verlangt der Lehrplan nur die Synthese.
Die Evaluation ist jedoch wichtig, um die vor allem in der Softwaretechnik vielfältig vorhandenen Lösungen zu vergleichen und zu bewerten. Durch die Erfahrungen, die der Lernende dabei gewinnt, werden die Modelle in Zukunft besser. Dies bedeutet, dass durch die Erweiterung auf die Evaluation die Problemlösefähikeit und die Modellbildung verbessert werden.

Neben den bisher vorgestellten kognitiven Lernzielen sind im Unterricht auch affektive und pragmatische anzustreben. Auch wenn in dieser Arbeit der Schwerpunkt auf den kognitiven Lernzielen liegt, sollen die anderen kurz erläutert werden, um zu zeigen, dass diese wichtigen Lernziele[15] auch im Informatikunterricht verfolgt werden sollen.
Das Konzept (siehe Kapitel 3.6 und 3.7) selbst nimmt keinen Bezug auf die affektiven und pragmatischen Lernziele. Eine Ausnahme bildet die Erwähnung von Lernformen wie Partner- oder Gruppenarbeit, ohne jedoch dabei auf die möglichen affektiven Lernziele, welche dabei verfolgt werden könnten, Bezug zu nehmen.

Die affektiven wurden u.a. wiederum von Bloom [Bl75] bearbeitet, dabei werden die Ziele nach dem Grad der Verinnerlichung der Werte gestuft:

aufmerksam werden/beachten:	Ein Werteproblem wird erkannt (Beispiel: Schüler wird auf kooperatives Verhalten zu seinen Mitschülern hingewiesen).
reagieren:	Auf das Werteproblem wird mit einer Reaktion geantwortet (Bei-

15 v.a. die affektiven Lernziele

	spiel: Schüler reagiert mit der Frage, in welcher Situation er kooperativer sein soll).
werten:	Das Werteproblem wird einer Bewertung unterzogen (Beispiel: Der Schüler wertet selbst sein Verhalten).
organisieren:	Die Haltung wird in den inneren Wertekatalog aufgenommen (Beispiel: Der Schüler nimmt sich vor, sein Verhalten zu ändern).
Wertestruktur:	Die Haltung ist verinnerlicht und ergibt unbewusst entsprechende Reaktionen (Beispiel: Der Schüler hat das Verhalten verinnerlicht, so dass er kooperativ ist).

Um Haltungen zu vermitteln müssen die Schüler eine Gelegenheit für sozialen Umgang untereinander und zu anderen erhalten. Besonders in Partner- und Gruppenarbeit sind diese affektiven Lernziele deswegen zusätzlich zu vermitteln.

Die pragmatischen Lernziele, wie sie z.B. von Dave [Da68] angegeben werden, beziehen sich hier vor allem auf die Codierung und den Umgang mit dem Programmcode. Sie sind nach der Koordination gestuft:

Imitation:	Eine Tätigkeit wird nachgemacht (Beispiel: Ein Programm wird abgeschrieben).
Manipulation:	Die Tätigkeit wird manipuliert (Beispiel: Das Programm wird spielerisch variiert).
Präzision:	Die Manipulationen werden gezielt eingesetzt (Beispiel: Das Programm wird gezielt manipuliert, so dass sich ein geplantes Ergebnis ergibt).
Handlungsgliederung:	Ein größerer Handlungsablauf wird geplant und ausgeführt (Beispiel: Es wird ein neues Programm entworfen und umgesetzt).
Naturalisierung:	Die Arbeitsabläufe haben sich automatisiert, so dass nicht mehr über alle Einzelheiten nachgedacht werden muss.

Eine weitere Anwendung für pragmatische Lernziele in der Informatik ist die Verwendung der Diagramme (z.B. Klassendiagramm, siehe Abbildung 13). Zuerst werden die Diagramme abgezeichnet (*Imitation*), dann werden sie manipuliert, indem z.B. die Anordnung der Klassen im Klassendiagramm nicht mehr der Vorlage (z.B. der Tafel) entspricht. Im nächsten Schritt (*Präzision*) werden die Klassen so angeordnet, dass sich logische Bereiche bilden, z.B. alle Klassen, welche ähnliche Aufgaben haben, nah beieinander liegen. Die *Handlungsgliederung* und *Naturalisierung* wird erreicht, wenn dieses logische Zusammenfassen intuitiv erfolgt.

Die affektiven und pragmatischen Lernziele stehen nicht im Mittelpunkt des Konzepts, wenn auch an den entsprechenden Stellen im Unterricht darauf geachtet werden sollte.

Vor allem die affektiven Ziele können vom Lehrenden bei Partner- und Gruppenarbeiten immer angestrebt werden. Dazu sollte er auf soziales Verhalten untereinander achten und dies evtl. auch immer wieder thematisieren.

3.6 Das grobe Konzept

Nachdem die Inhalte und die Lernzieltaxonomien für den objektorientierten Softwareentwurf erarbeitet wurden, steht in diesem Kapitel der zeitlich-inhaltliche Aufbau der Gesamteinheit im Vordergrund. Im Mittelpunkt steht der handlungsorientierte Aufbau, wie er in den didaktischen und methodischen Leitgedanken (siehe Kapitel 3.1) bereits erwähnt wurde, also entgegengesetzt zum realen Softwareentwurf. Des Weiteren steht die Darstellung dieses Aufbaus im Zentrum.

Das grobe Konzept soll für die Sekundarstufe II und den tertiären Bereich gelten. Wie bereits erwähnt, besteht jedoch zwischen den beiden Bereichen unter anderem ein Unterschied darin, dass im tertiären Bereich höhere Lernzieltaxonomien angestrebt werden. Deswegen spielen im Entwurf des groben Konzepts hauptsächlich die Leitgedanken eine Rolle. Genauere Lernziele kommen erst im detaillierten Konzept (siehe Kapitel 3.7) stärker zum Tragen, da dann nur noch für die Sekundarstufe II entwickelt wird.

Bevor auf den Aufbau eingegangen wird, soll zuerst eine Möglichkeit vorgestellt werden, wie die Lernaktivitäten mit Hilfe eines genormten Diagramms der Softwaretechnik, dem Aktivitätsdiagramm, dargestellt werden können. Das Aktivitätsdiagramm ist das in der UML spezifizierte Diagramm für die Darstellung von Prozessketten [OMG00]. Damit lassen sich Nebenläufigkeiten und alternative Schritte leicht darstellen. Eine kurze Erklärung der genormten Notation zeigt Abbildung 7.
Aktivitäten werden mit einem abgeflachten Oval dargestellt. Diese können in einem weiteren Aktivitätsdiagramm noch genauer in Unteraktivitäten beschrieben werden.
Die Linien mit Pfeilenden geben die gerichteten Verbindungen der Aktivitäten an. Waagrechte Balken symbolisieren Synchronisationspunkte. Die abgehenden Linien können erst betreten werden, wenn alle eingehenden Linien erreicht sind, z.B. können in Abbildung 8 die Aktivitäten *weitere Entwurfsmethoden* und *vertiefte Konzepte* erst begonnen werden, wenn die Aktivitäten *Beziehungen* und *Generalisierung* erledigt sind. Diese können dann (quasi) parallel ausgeführt werden.
Eine Raute steht für einen optionalen Weg mit Aktivitäten, welche nicht unbedingt oder nur unter bestimmten Bedingungen ausgeführt werden müssen.
Ein kleiner ausgefüllter Kreis und ein ausgefüllter Kreis mit umgebendem Kreis sind der Start- bzw. End-Punkt des gesamten Prozesses.

Auch Unterricht lässt sich als ein Prozess (Lernprozess) vorstellen. Nebenläufige Prozesse[16] sind dabei nicht direkt möglich, da der Lernende sich im selben Augenblick nur einem Thema widmen kann. Nebenläufigkeit wird dabei durch eine Serialsierung, d.h. eine hintereinander ablaufende Struktur der Prozesse erreicht, wobei die Reihenfolge beliebig ist. Dies entspricht auch der Realisierung von nebenläufigen Aktivitäten in Geschäftsprozessen, in denen z.B. nur ein Sachbearbeiter für die Abarbeitung der parallelen Prozesse zuständig ist.

16 In informationstechnischen Systemen können nebenläufige Prozesse häufig auch zeitlich parallel abgearbeitet werden, um z.B. Zeit zu sparen.

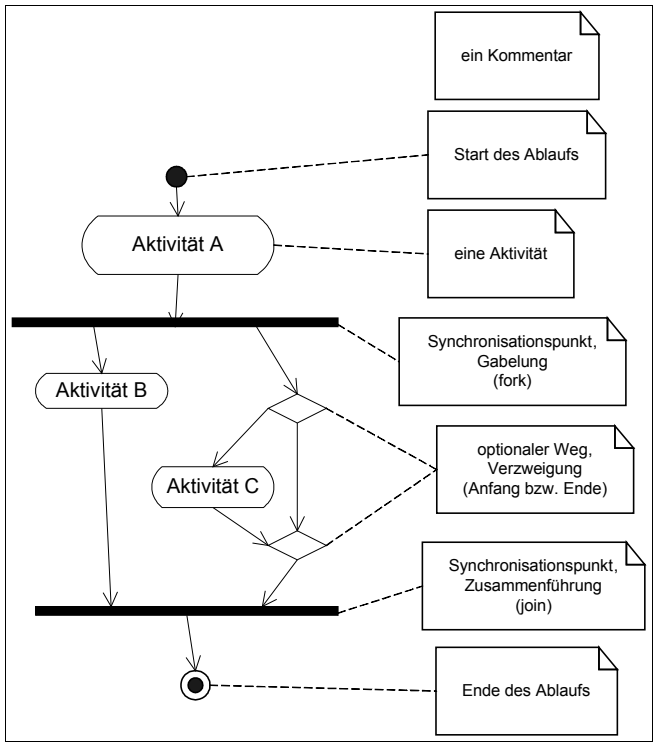


Abbildung 7. Erläuterungen zum Aktivitätsdiagramm

Aus diesem Grund eignet sich ein Aktivitätsdiagramm recht gut zur Strukturierung von Unterricht und bietet so eine anschauliche Darstellung des Ablaufs mit möglichen Alternativen und Verzweigungen. Dies sollen auch die folgenden Diagramme, in denen das hier erstellte Konzept dargestellt wird, zeigen.

Einen Überblick über die Reihenfolge der gesamten Unterrichtseinheit bietet das Aktivitätsdiagramm in Abbildung 8. Die einzelnen Aktivitäten werden dann später in Unteraktivitäten genauer betrachtet.
Meist werden in den Kursen für den objektorientierten Softwareentwurf die objektorientierten Grundkonzepte, wie z.B. die Begriffe *Objekt*, *Klasse* oder *Vererbung* aus vorhergehenden Kursen vorausgesetzt. Dies ist auch der richtige Weg, zuerst müssen die Grundkonzepte geklärt sein. Im hier vorgestellten Konzept werden aber bereits hier einfache Entwürfe entwickelt, um die Grundkonzepte verständlich zu machen und Handlungsorientierung zu erreichen.
Nach dem Einstieg über die Klärung der Begriffe *Objekt und Klasse* erfolgt eine weitergehende Betrachtung über Beziehungen (Assoziationen) und über die Generalisierung. Die Reihenfolge in der Wahl der beiden Abschnitte ist freigestellt (Nebenläufigkeit).
In der Aktivität *Beziehungen* werden die bisher einzeln betrachteten Klassen verbunden, sie gehen nun Assoziationen ein. Es wird darauf geachtet, dass jede Klasse eine ihr zugewiesene Aufgabe zu erfüllen hat. Wie bereits in der vorangegangenen Aktivität werden hier auch Entwürfe gestaltet. Besonders wichtig sind dabei die Designrichtlinien, mit deren Hilfe die Verteilung der Aufgaben und damit der Attribute und Methoden auf die Klassen vereinfacht werden. Es werden bereits hier die Lernzieltaxonomien *Synthese* und *Evaluation* angestrebt (siehe Kapitel 3.5).

Auch die *Generalisierung* verbindet Klassen untereinander. Allerdings handelt es sich hier nicht um eine Beziehung, sondern um eine Zusammenfassung von Klassen in Form von Generalisierung, bzw. in der Gegenrichtung betrachtet einer Spezialisierung von Aufgaben. Die Unterklasse (Subklasse) besitzt dieselbe Aufgabe wie die Oberklasse (Superklasse) aber in einem stärker spezialisierten Sinn, wie z.B. die Subklasse *LuftFahrzeu*g eine Spezialisierung der Superklasse *Fahrzeug* darstellt, indem hier der Ort der Fortbewegung auf die Luft spezialisiert wird.
Auch hier wird entworfen, wobei auf die richtige Wahl der Generalisierung Wert gelegt wird: Wann sollte generalisiert werden und wie erkennt man eine gute Generalisierungsstruktur?

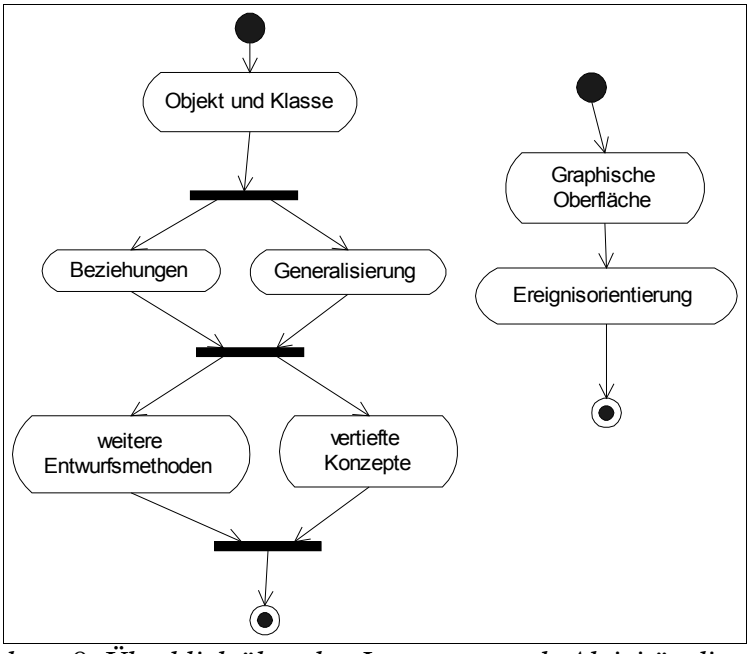


Abbildung 8. Überblick über den Lernprozess als Aktivitätsdiagramm

Nachdem die Einheiten (Aktivitäten) *Objekt und Klasse*, *Beziehungen* und *Generalisierung* bearbeitet sind, ist das Grundgerüst des objektorientierten Softwareentwurfs erledigt. Nun steht der Weg frei für weitere Entwurfsmethoden und vertiefte Konzepte.
Bei den weiteren Entwurfsmethoden stehen weitere Verfahren zum Entwurf im Mittelpunkt. So kann hier z.B. das Zustandsdiagramm für das Design zustandsbehafteter Abläufe, die Palette der Patterns[17], welche Musterlösungen für oft vorkommende Probleme bieten, oder weitere Methoden zum Finden von Klassen und deren Beziehungen untereinander vermittelt werden.
Hier ist man im Aufbau der Gesamteinheit an der Stelle angelangt, welche im realen Entwurfsprozess am Anfang steht: Der Formulierung der Anforderungen an das Programm, z.B. über Use Cases. Nun sollte den Lernenden die Objektorientierung so vertraut sein, dass dieser Schritt für sie verständlicher ist, als dies am Anfang der Einheit möglich gewesen wäre.
Als Abschluss der Gesamteinheit bietet sich dann auch eine Gesamtsicht des Entwurfsprozesses mit allen Verfahren und Methoden an.

17 Begriff: siehe Glosar (Anhang E)

Die vertieften Konzepte beziehen sich auf weitere Konzepte der objektorientierten Programmierung und nicht auf Entwurfsverfahren. Sie sind meist nur für spezifischere Probleme nötig. Zum Teil können sie auch im Sinne von Handlungsorientierung in die anderen Teile integriert werden, um die für die Vermittlung der dort genannten Inhalte gewählten Beispiele umzusetzen. Allerdings dürfen diese vertieften Konzepte dabei nicht im Mittelpunkt stehen.

Entsprechendes gilt auch für die Aktivitäten *graphische Oberfläche* und *Ereignisorientierung* in Abbildung 8. Vor allem die Aktivität *graphische Oberfläche* dient primär der Motivation und steht in diesem Konzept inhaltlich nicht im Vordergrund. Die Ereignisorientierung stellt eine Neuerung dar, da nun der Anwender eine größere Freiheit in der Reihenfolge der Abarbeitung besitzt. Sie wird aber nicht nur im objektorientierten Softwareentwurf verwendet.

Anschließend sollen nun die oben genannten Aktivitäten genauer strukturiert werden. Auch dies erfolgt wiederum mit Hilfe eines Aktivitätsdiagramms.

3.6.1 Objekt und Klasse

Der Ablauf dieser Einheit ist Abbildung 9 zu entnehmen. Als Einstieg wird eine kleines Programm gewählt, welches analysiert werden soll. Diese Grundidee stammt dabei von Magenheim [Ma99]. Dieses Programm hat den Vorteil, dass die Lernenden direkt den Code untersuchen, mit ihm experimentieren können und so einen spielerischen Einstieg finden. Die Objekte werden unmittelbar klar.
Im Gegensatz zu Magenheim ist das Programm aber so angelegt, dass es relativ klein ist und zum Teil bereits Hilfen für die Erweiterung bietet. Des Weiteren wird es später mit den erworbenen Kenntnissen Stück für Stück erweitert.

Als Hilfsmittel zur Programmanalyse wird sofort das Objektdiagramm und im nächsten Schritt das Klassendiagramm eingeführt. Allerdings besteht an dieser Stelle noch keine Verbindung zwischen den Klassen. Der Schwerpunkt liegt auf dem Finden der Klasse, ausgehend von den Objekten und deren Attributen und Methoden, also im inneren Aufbau der Klasse. An dieser Stelle ist auch der richtige Augenblick, um evtl. auf Referenzen oder Zeiger (je nach Programmiersprache) einzugehen, da hiermit oftmals die erzeugten Objekte erreicht werden. Sollte den Lernenden die Unterscheidung zwischen statischen und dynamischen Variablen noch nicht bekannt sein, so ist dies der Zeitpunkt diese einzuführen. Dadurch wird implizit die Verbindung zwischen Code und den Diagrammen hergestellt, damit der Lernende nicht die Orientierung verliert. Weiterhin wird auch der Übergang zum Softwareentwurf erleichtert, da dann nicht mehr im Code, sondern nur noch mit Hilfe der Diagramme gearbeitet werden soll. Zu diesem Zeitpunkt sollten die Lernenden dann die Diagramme verinnerlicht haben.

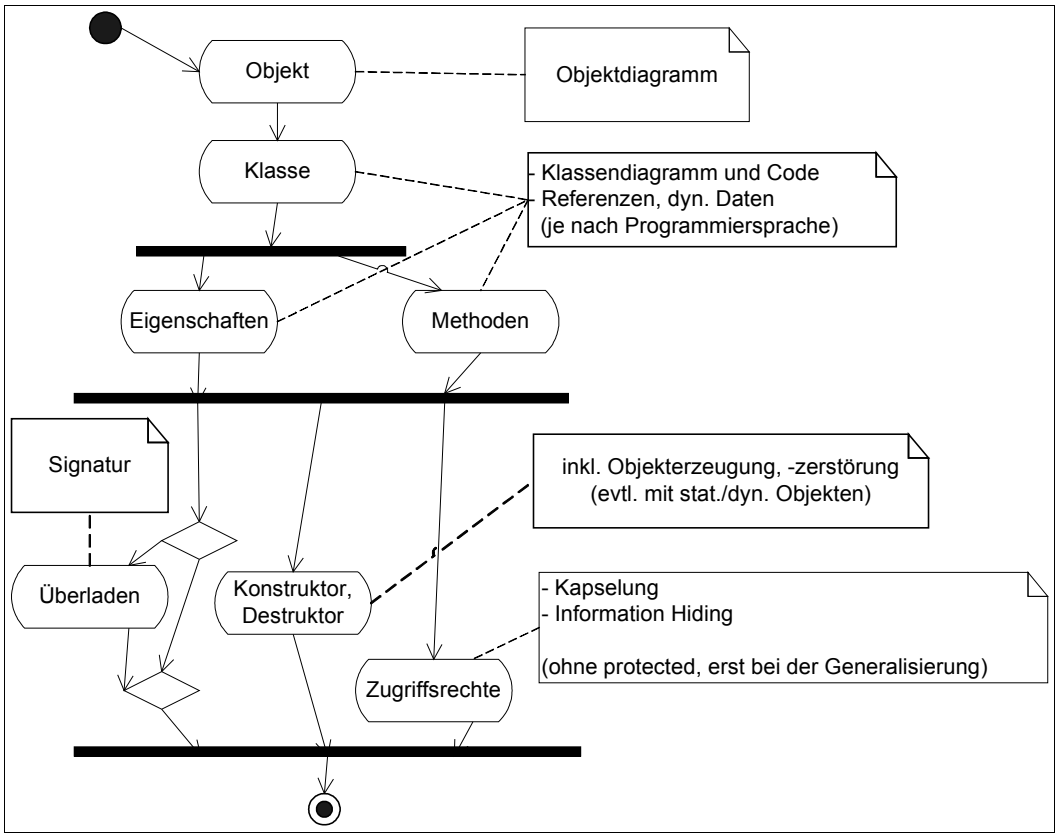


Abbildung 9. Die Details zur Aktivität "Objekt und Klasse" als Aktivitätsdiagramm

Die Grundidee von Magenheim wurde also in der Art erweitert, dass das verwendete Programm so einfach ist, dass die Schüler das Modell in Form des Klassendiagramms intuitiv richtig entwickeln und dass das so gefundene Modell quasi als roter Faden für den weiteren Unterricht, Stück für Stück weiterentwickelt wird und die Schüler dabei die weiteren Grundprinzipien bzw. Verfahren des objektorientierten Softwareentwurfs kennen lernen.
Im Rahmen dieser Einheit sollte man anschließend auf die besonderen Methoden des Konstruktors und Destruktors eingehen, da sie eine wichtige Rolle bei der Initialisierung der Objekte bzw. beim Aufräumen spielen. Damit verbunden ist direkt auch die programmtechnische Umsetzung der Erzeugung bzw. Vernichtung von Objekten. Auch die Zugriffsrechte spielen eine wichtige Rolle. Erst mit ihnen kann die innere Struktur der Klasse vor der Umgebung verborgen werden (Kapselung). Allerdings kann das Zugriffsrecht *protected* noch nicht behandelt werden, da die Generalisierung, welche dazu bekannt sein muss, erst später Thema ist.
Optional kann hier noch das Überladen[18] von Methoden behandelt werden. Für das grundlegende Verständnis der Objektorientierung ist dies aber nicht zwingend notwendig. In der Programmierrealität wird dieses Prinzip jedoch oft verwendet wie auch die meisten objektorientierten Programmbibliotheken zeigen.

3.6.2 Beziehungen

Bereits in den nächsten Schritten (*Beziehung*, *Generalisierung*) werden die ersten Softwareentwürfe gestaltet. Die Reihenfolge spielt im Grunde genommen hier keine Rolle, wes-

18 Begriff: siehe Glosar (Anhang E)

halb diese im Aktivitätsdiagramm (siehe Abbildung 8) als parallele Aktivitäten dargestellt sind.

Im Bereich der Beziehungen werden zuerst die Grundbegriffe wie *Assoziation*, *Beziehungsnahme*, *Rolle* und *Kardinalität* behandelt (siehe Abbildung 10). Dies erfolgt wiederum am bekannten Programm, beginnend mit der Assoziation, welche die Basis für die oben erwähnten Begriffe bildet. Das bereits bekannte Programm hat den Vorteil, dass die Lernenden am Beispiel lernen und dabei auch weitere wichtige Gesichtspunkte bei der Programmierung vertiefen: Eine gute Dokumentation erleichtert die Einarbeitung in das Programm, Beispiele für die Strukturierung von Klassen etc.
Kardinalitäten helfen einerseits bei der Problemanalyse (OOA) das Problem genauer einzugrenzen (Sollen Autos mit 3 Rädern berücksichtigt werden oder nur mit 4 Rädern?). Andererseits kann beim Design (OOD) schon abgeklärt werden, ob ein einzelnes Attribut (Kardinalität 1), ein statisches Array (endliche bekannte Kardinalität) oder bei der Kardinalität *viele* ein dynamisches Array eingesetzt werden soll.

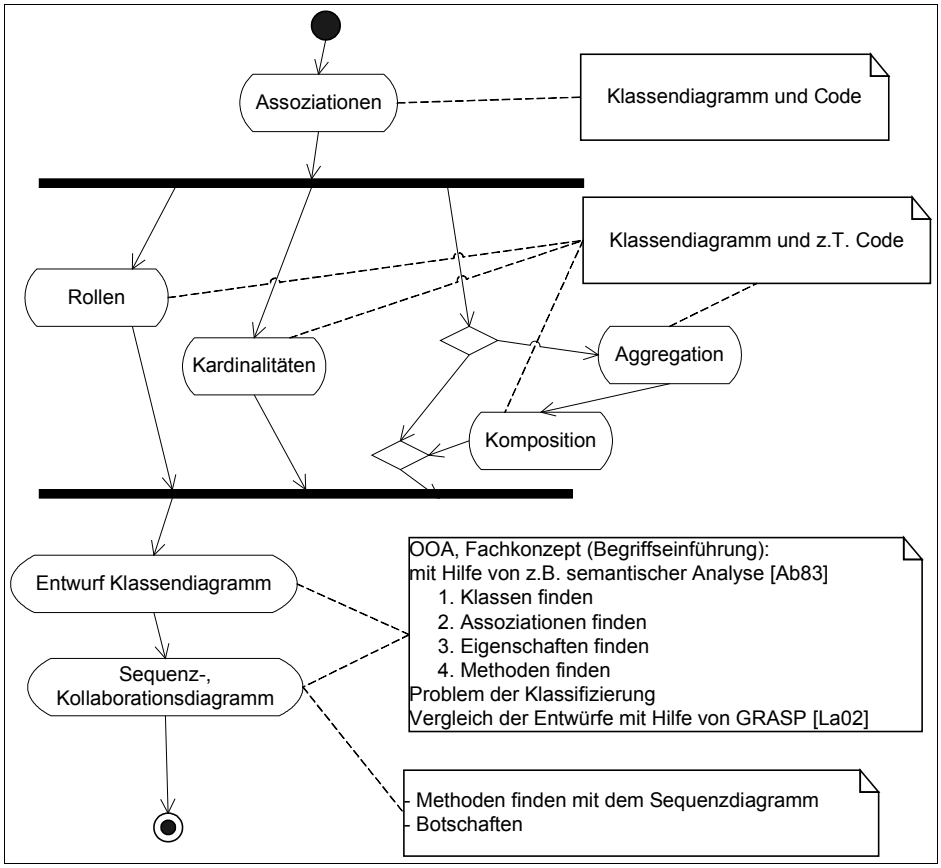


Abbildung 10. Die Details zur Aktivität "Beziehungen" im Aktivitätsdiagramm

Auch Rollen helfen das Problem genauer einzugrenzen und ersetzen manches Mal eine Vererbung. Eine Spezialisierung muss nicht unbedingt per Vererbung umgesetzt werden. Manchmal ist eine Beziehung mit Rollen günstiger, da ein vererbtes Objekt zwar unter verschiedenen Gestalten auftreten kann (Polymorphie, s.u.), aber die eigene Identität nicht ändern kann. So kann ein einzelnes Objekt aus einer Menge von gleichartigen Objekten (aus derselben Klasse) nur vorübergehend eine zusätzliche Eigenschaft benötigen. Allen diesen

gleichartigen Objekten diese Eigenschaft zu geben ist bei einer entsprechenden Größe der Menge nicht sinnvoll. Eine per Vererbung zugewiesene zusätzliche Eigenschaft kann das Objekt nicht verlassen.
Ein Beispiel dieser Problematik ist die Eigenschaft einer demokratisch gewählten Person (siehe Klassendiagramm in Abbildung 11). Nur während der Wahlperiode besitzt sie zusätzliche Eigenschaften, wie z.B. das Rederecht im Parlament. Wird diese Person abgewählt, verliert sie diese Eigenschaften, ohne aber als Objekt zu verschwinden. Als Modell eignet sich hier eine Klasse *Person* und eine Klasse *Parlamentssitz*, wobei nur die *Person(en)* in der Rolle des *Abgeordneten* einen Parlamentssitz mit dem entsprechenden Rederecht erhält. Alle anderen Personen besitzen diese Eigenschaft nicht. Wird ein Parlamentarier abgewählt, so verliert er diese Rolle, bleibt aber weiterhin *Person*. Mit Vererbung ist dieses Verhalten nicht zu modellieren.

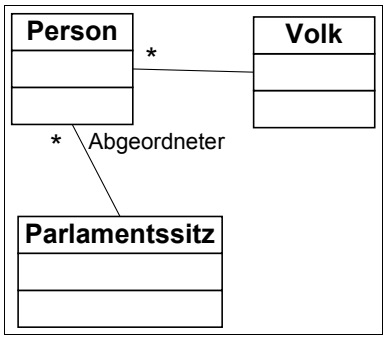


Abbildung 11. Beipiel eines Klassendiagramms zur Verwendung einer Rolle (Erläuterungen siehe Text)

Optional können hier zusätzlich die Konzepte *Aggregation* und *Komposition* eingeführt werden. Sie schärfen das Denken, indem genau überlegt werden muss, ob es sich bei der Assoziation um eine Teil-Ganzes-Beziehung handelt und ob sie von existentieller Natur ist. Ein Beispiel ist eine Klasse, die das Teil (z.B. Finger) repräsentiert und existentiell von der Klasse, welche das Ganze darstellt (z.B. Hand) abhängt. Nur dann handelt es sich um eine Komposition. Allerdings sind dies sehr spezielle Konzepte, sodass sie auch weggelassen werden können.

Es werden die Konzepte im Code und im Klassendiagramm erarbeitet. Hat man entsprechende CASE-Tools im Einsatz, kann auch das Reengineering des Codes in das Klassendiagramm gezeigt werden: Der Zusammenhang vom Code zum Klassendiagramm wird unmittelbar bewusst. Im Klassendiagramm ist dann der Programmaufbau leichter ersichtlich, der Vorteil von UML wird unmittelbar verstanden. Allerdings muss man als Lehrender dabei darauf achten, dass durch die Möglichkeit des Reengineerings nicht der Weg vom Programm zum Klassendiagramm von den Lernenden gewählt wird.

Nachdem nun wiederum die Konzepte geklärt sind, werden diese in den Entwurf integriert. Es wird eine objektorientierte Analyse (OOA) durchgeführt, d.h. der Schwerpunkt der Arbeit liegt auf der Frage, welches Problem gelöst werden soll und nicht wie es gelöst werden soll. Es werden Hilfen angeboten, wie man von der Problemstellung zu einem Modell gelangt.
Die Problemstellung liegt in Form einer textuellen Beschreibung vor (z.B. als Dialog zwischen Entwickler und Kunde). Folgender Weg zur Lösung bietet sich an:

1. Klassen finden (z.B. Substantive im Text, Gattungsnamen, Eigennamen) (Methode nach Abbott [Ab83]).
2. Assoziationen finden (z.B. Beziehungen zwischen den Substantiven)

3. Eigenschaften finden
4. Methoden finden (z.B. Verben im Text oder über das Sequenzdiagramm)

Bei der Erstellung des Klassendiagramms (Lernzieltaxonomiestufe: Synthese) und bei der Untersuchung dieser (Lernzieltaxonomiestufe: Evaluation) helfen die Designrichtlinien nach C. Larman (GRASP [La02], siehe Kapitel 3.4). Deswegen sollten sie an dieser Stelle eingeführt werden.

Das Finden von Methoden wird durch die Analyse einzelner Szenarien erleichtert. An dieser Stelle hilft das Sequenz- bzw. Kollaborationsdiagramm bei der Suche, da vor allem das Sequenzdiagramm den zeitlichen Ablauf eines Szenarios gut abbilden kann. Aus den Botschaften, die dabei von Objekt zu Objekt versandt werden, ergeben sich direkt die Methoden der Klassen. Aus diesem Grund wird es hier eingeführt und das Erlernte sofort umgesetzt.

Sehr gut eignet sich an dieser Stelle auch, auf das Problem der Klassifizierung einzugehen, wie es Booch in seinem Buch in einem praktischen Beispiel darlegt ([Bc94, S.194f], siehe Anhang B.6). Es wird deutlich, dass die gefundene Klassenzuordnung abhängig von der Problemstellung ist. Eine KFZ-Verwaltung für eine KFZ-Werkstatt wird anders aussehen als für eine Zulassungsstelle.

In dieser Einheit wird der Bereich der Grundkonzepte mit dem Bereich des Softwareentwurfs verzahnt. Dieser Bereich besitzt einen hohen Bildungswert, da die reale Welt in ein Modell abgebildet wird. Dabei treten typischerweise die gleichen Probleme auf: Es müssen Vereinfachungen eingeführt werden und es gibt viele mögliche Lösungen mit ihren Vor- und Nachteilen. Die von den Lernenden gefundenen Lösungen sollen von ihnen vorgestellt und mit allen diskutiert werden, um diese Aspekte zu verdeutlichen. Dabei helfen die Designrichtlinien nach C. Larman (s.o.).
Solche Probleme sind typisch für alle technischen Disziplinen, woraus direkt der hohe Bildungswert folgt.

3.6.3 Generalisierung

Die Generalisierung lässt sich wieder im Programmcode und im Klassendiagramm zeigen. Das den Lernenden bekannte, nun etwas erweiterte Programm soll ausgebaut werden, um in dieses Thema einzuführen. Den Ablauf der Einheit zeigt Abbildung 12.

Nachdem die Grundkenntnisse zur Generalisierung vermittelt sind, erfolgt wieder ein Entwurf. Hier wird im Gegensatz zu den Beziehungen ein Design (**O**bject **O**riented **D**esign) entwickelt. Der Schwerpunkt liegt nun darauf, wie das Problem zu lösen ist. Auch dabei erfolgt vorher die Vermittlung einer methodischen Unterstützung als Hilfe für die Lernenden. Danach erfolgt die Codierung des Entwurfs (**O**bject **O**riented **P**rogramming).

Anschließend wird das Überschreiben von Methoden, die Polymorphie und optional die Mehrfachvererbung bzw. die Interfaces eingeführt.

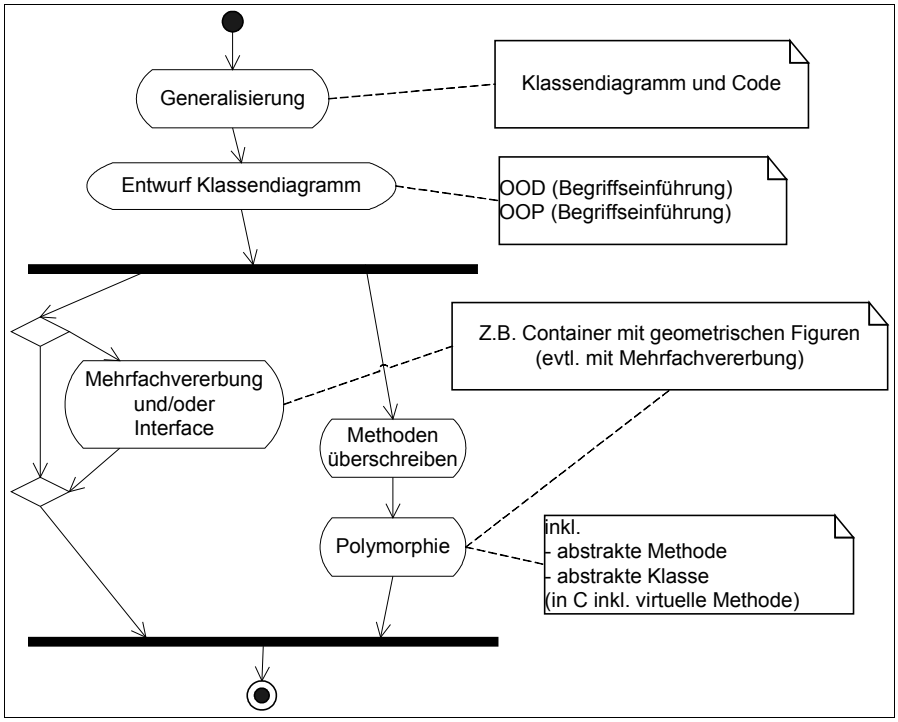


Abbildung 12. Die Details zur Aktivität "Generalisierung" im Aktivitätsdiagramm

Das besondere an der Polymorphie ist die Eigenschaft, dass erst zur Laufzeit des Programms festgelegt wird, welche Methode verwendet wird. Unterschiedliche Klassen mit allerdings derselben Superklasse können damit verwaltet werden. Aus Sicht der aufrufenden Methode wird eine Methode der Superklasse aufgerufen, ausgeführt wird jedoch die Methode der Subklasse, welche die Methode der Superklasse überschrieben hat. *Überschreiben* nennt man das Verwenden desselben Methodennamens inkl. Parameterliste in einer Subklasse. Hiermit wird erst das Prinzip der Container-Klassen möglich. Diese sind Speicherorte für Objekte der Superklasse, wobei auch Subklassen im Sinne der Polymorphie darin abgelegt werden können.

In diesem Zusammenhang spielen auch abstrakte Methoden, also Methoden ohne Programmrumpf, und abstrakte Klassen, von denen keine Objekte erzeugt werden können, eine Rolle. Abstrakte Methoden erzwingen in den Subklassen eine Implementierung dieser Methode, um die Subklasse verwenden zu können, so dass die Klasse nicht mehr abstrakt ist.

Wegen ihres anspruchsvollen Charakters sollte besonders die Polymorphie handlungsorientiert eingeführt und in eine entsprechende Problemstellung verpackt werden. Besonders gut eignet sich hier eine Problemstellung zu geometrischen Figuren: Mehrere geometrische Figuren (Kreise, Rechtecke, Dreiecke, Schablonen) sollen gezeichnet bzw. die Gesamtfläche der Figuren berechnet werden (Klassendiagramm siehe Abbildung 13). Hier ist des Weiteren auch die Umsetzung des Designs in Programmcode wichtig, da dadurch die Besonderheiten der Polymorphie – erst zur Laufzeit wird die zu verwendende Methode festgelegt – klarer zu Tage treten.

Die Mehrfachvererbung, bei der eine Klasse mehrere Superklassen besitzt und die Interfaces, bei der als Superklasse eine Klasse mit ausschließlich abstrakten Methoden verwendet wird, sind nur optional. Beide Konzepte sind sprachabhängig, so wird die Mehrfachvererbung z.B. nicht von Java und das Konzept des Interfaces nicht von C++ umgesetzt.

Sollte die Mehrfachvererbung thematisiert werden, so kann dies unter Verwendung der Figur eines Rechtecks mit ausgeschnittenem Kreis (Schablone) im obigen Projekt eingegangen werden (siehe auch Abbildung 13).

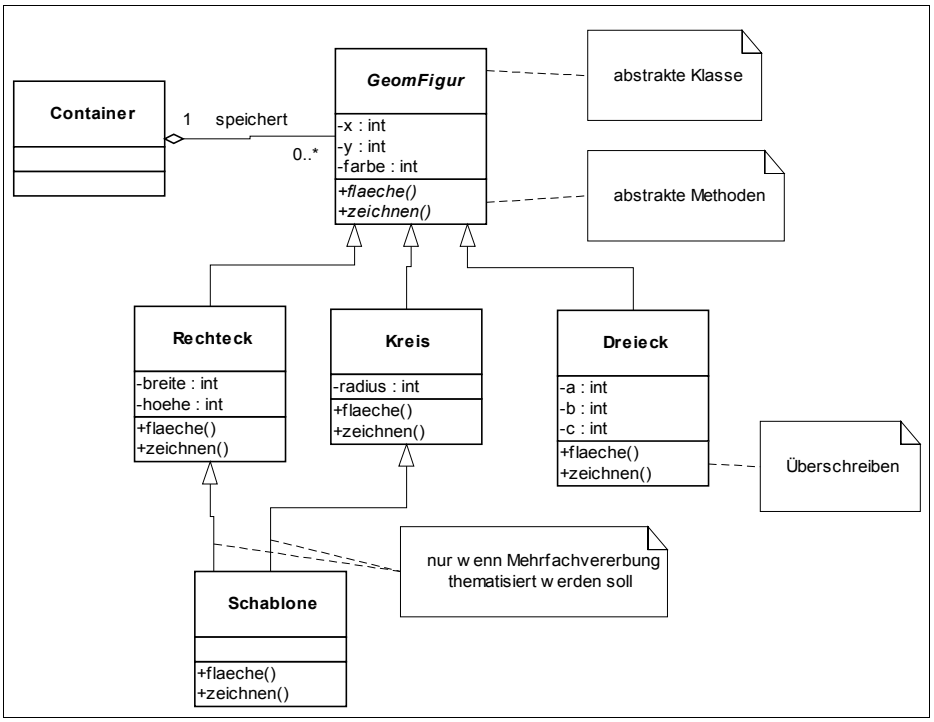


Abbildung 13. Das Klassendiagramm für das Problem der geometrischen Figuren

3.6.4 Weitere Entwurfsmethoden

Im Grunde sind an dieser Stelle alle grundlegenden Gedanken des objektorientierten Softwareentwurfs entwickelt. Es folgen nun weitere Konzepte und weitere Entwurfsmethoden (Aktivität *vertiefte Konzepte* bzw. *weitere Entwurfsmethoden* in Abbildung 8). Deswegen sind die meisten Unteraktivitäten optional (siehe Abbildung 14 bzw. Abbildung 15).
Eine Ausnahme bildet nur das Suchen nach Use Cases, da dies das wichtige Thema der Anforderungsanalyse anspricht und der Gesamtüberblick über den Ablauf beim objektorientierten Softwareentwurf.
Anwendungsfälle (Use Cases) sind in der Zwischenzeit wichtige Hilfsmittel für die Problemanalyse. Viele Softwareentwurfsmethoden haben den Use Case als den Ausgangspunkt [Ja92][La02][Kr99]. Deswegen sollte dieser auf jeden Fall eingeführt werden. Auch der anschließende Gesamtüberblick über das methodische Vorgehen, um vom Problem zur Lösung zu kommen, beginnt mit einem Use Case.
Einige der optionalen Aktivitäten können auch schon zu einem früheren Zeitpunkt eingebracht werden, sofern dies ein handlungsorientiertes Beispiel erfordern würde, wie z.B. der Entwurf des Zustandsdiagramms. Mit Hilfe dieses Diagramms, welches bereits lange vor Einführung der Unified Modeling Language (UML) verwendet wurde, lassen sich zustandsbehaftete Probleme lösen. Um das Verständnis zu vertiefen, sollte hier ein Problem exemplarisch nach der Modellierung in Programmcode umgesetzt werden.

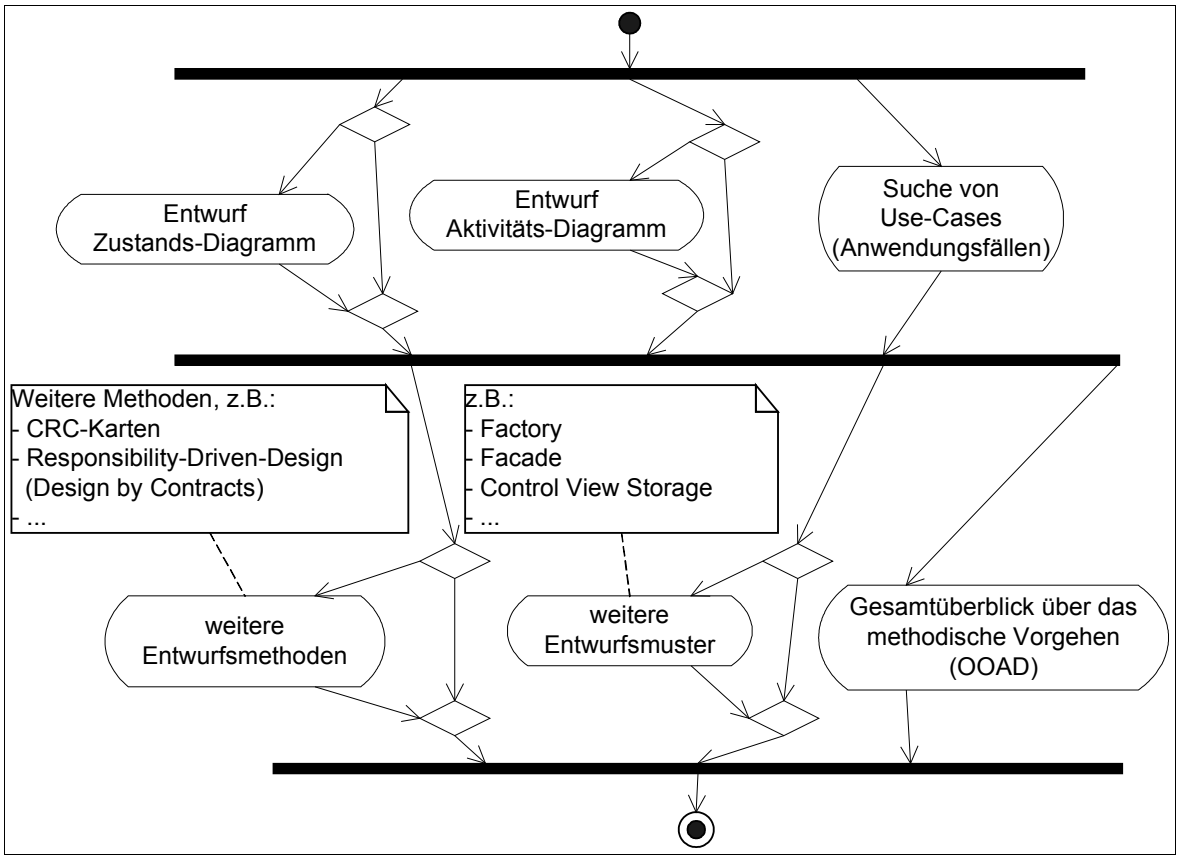


Abbildung 14. Die Details zur Aktivität "weitere Entwurfsmethoden" im Aktivitätsdiagramm

Andere optionale Aktivitäten sind zu einem früheren Zeitpunkt jedoch noch ungeeignet. Zu nennen ist hier der Entwurf des Aktivitätsdiagramms, welches zusammen mit den Use Cases zur Anforderungsanalyse gehören oder die weiteren Entwurfsmuster. Für letzteres sollte bereits eine fundierte Kenntnis der Konzepte und Methoden der Objektorientierung als auch eine gewisse Erfahrung vorhanden sein.
Aktivitätsdiagramme sind vor allem für die Beschreibung einzelner Anwendungsfälle (Use Cases) hilfreich. Längere Prozessketten mit ihren zum Teil auftretenden Nebenläufigkeiten können damit dargestellt werden.
Ansonsten werden einzelne Anwendungsfälle über textuell beschriebene Szenarien (siehe Abbildung 75, Anhang A.4.4) ähnlich einem Storyboard bei der Produktion von Filmen dargestellt. Im Mittelpunkt stehen dabei die sequentiell abfolgenden Aktionen eines externen Aktors mit den Reaktionen des Systems. Aus der unmittelbaren fachlichen Nähe zu den Use Cases folgt hier auch die zeitliche Nähe in der Vermittlung dieses Inhalts.
Die weiteren Entwurfsmethoden sollten nicht nach vorne verlagert werden, um die Lernenden zu Beginn nicht zu sehr zu verwirren. Allerdings kann anstatt dem hier gewählten Verfahren der textuellen Analyse zum Finden von Klassen (siehe oben) auch ein anderes zur Einführung verwendet werden.

Der Gesamtüberblick über die methodischen Verfahren entspricht gleichzeitig einer Wiederholung des gesamten Stoffs und sollte das letzte Thema der Gesamteinheit bilden. Dies lässt sich sehr gut von den Lernenden zusammenfassen und dann vortragen.
Als neues Element kommt hier das iterative Vorgehen hinzu, welches beim objektorientierten Softwareentwurf immer wieder betont wird [La02][Kr99][Ne02]. Im Mittelpunkt steht dieses Vorgehen beim Unified-Process [Kr99]. Es stammt von den Vätern (Booch, Rumbaugh, Jacobsen) der Unified Modeling Language (UML) und ist besonders auf die Vorgehensweise beim objektorientierten Softwareentwurf abgestimmt (siehe auch Kapitel 2.1). Dadurch ist es gut geeignet, exemplarisch als Management-Verfahren vorgestellt zu werden.

Nach der Vorstellung des Unified-Process sollte sich an diese letzte Einheit ein Projekt anschließen, in welchem die Prozessschritte von der Analyse, über das Design, die Implementierung und den Tests von den Lernenden dokumentiert werden. Ein iteratives Vorgehen wird sich dabei zwangsläufig einstellen.

3.6.5 Vertiefte Konzepte

Wie bei den erweiterten Methoden bereits erwähnt sind die Unteraktivitäten hier alle optional (Abbildung 15). Hier gilt für alle, dass sie auch früher verwendet werden können, sofern es die handlungsorientierten Beispiele verlangen. In diesem Konzept ist dies z.B. für die Container der Fall, welche bereits zusammen mit der Polymorphie im Projekt zum Zeichnen der geometrischen Figuren verwendet wurden. Allerdings sollte man bei dem Wunsch, noch tiefer in die objektorientierte Programmierung einsteigen zu wollen, nicht zu viele Konzepte in ein Beispiel packen. Dies könnte die Lernenden überfordern. In diesem Fall ist es besser, am Ende des Curriculums weitere Beispiele zur Vermittlung einzuführen, welche einerseits das bisher Gelernte wiederholen und vertiefen und andererseits auf die weiteren Konzepte besser zugeschnitten sind.

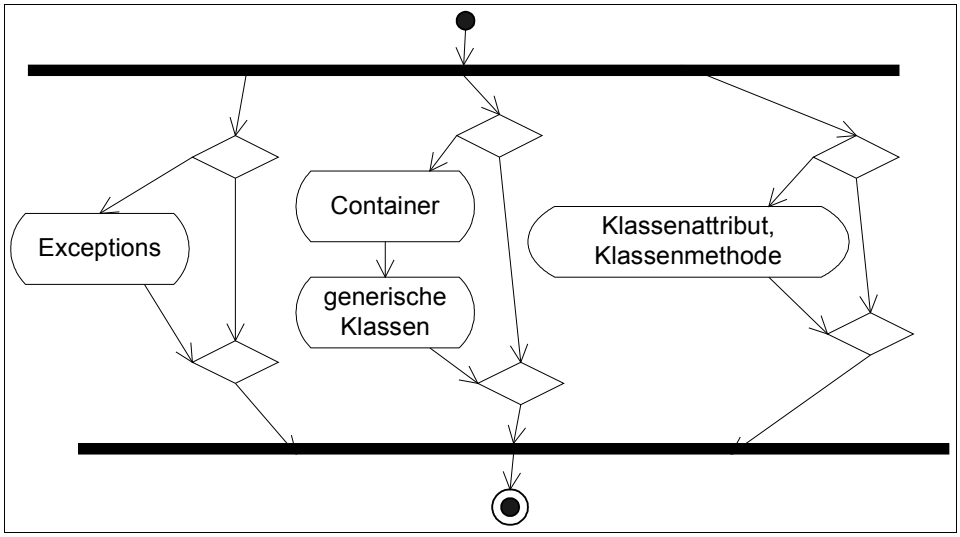


Abbildung 15. Die Details zur Aktivität "vertiefte Konzepte" im Aktivitätsdiagramm

Als vertiefte Konzepte sind hier vor allem die Exceptions, die Klassenattribute und Klassenmethoden und die Container zu nennen. Je nach Sprache spielen dabei auch generische

Klassen (Templates) eine Rolle. Allerdings gehören hier auch alle noch nicht behandelten Konzepte wie sie in Kapitel 3.4 (siehe auch Abbildung 5) genannt wurden hinzu.
Exceptions wurden für die objektorientierte Programmierung eingeführt. Sie werden verwendet, um Fehler abfangen zu können. Durch diese Möglichkeit können Fehler klassenspezifisch verarbeitet werden, was wiederum der objektorientierten Philosophie entspricht.
Klassenattribute sind Attribute, welche nicht einem einzelnen Objekt sondern allen Objekten einer Klasse zugeordnet sind. So ist es programmtechnisch möglich, z.B. die Anzahl der Objekte einer Klasse zu begrenzen (Entwurfsmuster Singleton: max. ein Objekt). Entsprechendes gilt für Klassenmethoden. Diese können verwendet werden, ohne dass ein Objekt der Klasse erzeugt werden muss.
Das Konzept der Container wurde bereits oben erklärt. Mit Hilfe von generischen Klassen (Templates) kann dieses Konzept erweitert werden. Damit ist es möglich, erst bei der Erzeugung von Containern den abzulegenden Datentyp zu bestimmen, indem eine spezielle Art von Parameter den Datentyp für vorher festgelegte Variablen angibt.
Diese hier vorgestellten Konzepte sind aber zu spezifisch, als dass sie in der gymnasialen Oberstufe einen zwingenden Inhalt darstellen. Deswegen werden sie im folgenden detaillierteren Konzept nicht verwendet.

3.7 Das Konzept im Detail

Im Anschließenden wird das Konzept detaillierter entworfen. Nun ist es aber nicht mehr möglich, auf die unterschiedlichen Anforderungen der Bildungsstufen einzugehen, da die Unterschiede nun zu groß sind. Des Weiteren soll das Konzept konkret umgesetzt werden, so dass mit den vorhandenen Rahmenbedingungen gearbeitet werden muss. Die Umsetzung soll im Rahmen des informationstechnischen Gymnasiums in Baden-Württemberg in Klasse 12 erfolgen und dort in mehreren Schulen evaluiert werden. Die Rahmenbedingungen sind dabei durch die Anforderungen der Sekundarstufe II, durch das Vorwissen der Schüler in Informationstechnik aus Klasse 11 und durch den Lehrplan im Fach Informationstechnik der Klasse 12 [It01] gegeben. Dort findet sich eine Zeitvorgabe von 45 Unterrichtsstunden – plus 15 Stunden für Vertiefungen und Leistungskontrolle, welche jedoch in diesem Konzept nicht verplant werden sollen – für die Lehrplaneinheit 5 *Objektorientierte Analyse und Design*.
Beim Entwurf des detaillierten Konzepts wird das grobe Konzept als Grundlage verwendet. Ganz bewusst bleibt bei diesem Entwurf für den Lehrenden genügend Freiraum. Dies ist nötig, damit er auf die Rahmenbedingungen der Klasse, wie z.B. soziales Umfeld, Klassengröße, soziales Verhalten in der Klasse aber auch auf eigene Stärken und Schwächen Rücksicht nehmen kann. Die zu verwendenden Sozialformen wie z.B. das Unterrichtsgespräch oder die Gruppenarbeit werden meist nicht explizit erwähnt, um dem Lehrenden hier Freiraum zu gewähren. Welche Möglichkeiten in den konkreten Unterrichtseinheiten bestehen, erschließt sich aber aus den an die Lehrenden ausgehändigten Unterlagen (siehe Anhang A). Die dort zu findenden Abbildungen können als Vorlagen für ein Tafelbild, für Arbeitsblätter oder direkt per Beamer oder Overhead verwendet werden.
Ein Schwerpunkt soll die Abfolge der Einheiten und Untereinheiten darstellen. Dies wird mit Hilfe von Leitfragen erreicht, welche den Ablauf darlegen sollen und evtl. auch die Schüler lenken können.
Wichtig sind jedoch auch Beispiele für die Motivationphasen, die Phasen der Erarbeitung und für die praktische Umsetzung des Gelernten in kleineren Arbeitsaufträgen bis zu Pro-

jekten. Zum Teil wurden dazu auch Arbeitsblätter inkl. passender Musterlösungen erstellt (siehe Anhang B.). Teilweise sollen auch didaktische, methodische oder fachliche Hinweise erfolgen, um die Lerninhalte zu vertiefen, einzuüben oder zu betonen.

Als roter Faden für die Schüler wird von einem kleinen, bestehenden Programm ausgegangen, welches die Schüler schrittweise erweitern sollen. Wie bereits im Grobkonzept erwähnt stammt die Grundidee der Dekonstruktion von Magenheim [Ma99]. Allerdings wird hier ausgehend von der Analyse des Programms dieses sofort erweitert und damit strukturiertes Entwerfen vermittelt. Im weiteren Verlauf der Stunden wird dieses Programm Stück für Stück erweitert und werden damit die Grundkonzepte der objektorientierten Programmierung eingeführt.
Den Schülern wird nur die Grundversion des Programms ausgeteilt, alle Erweiterungen sollen dann von ihnen vorgenommen werden. Als Hilfe, um die einzelnen Lernziele besser zu erkennen, wurden jedoch auch die einzelnen Schritte im Entwicklungsprozess des Programms an die Lehrer der Umsetzungsgruppe verteilt[19]. Die unterschiedlichen Stufen der Entwicklung (Versionen) sind im Anhang D mit einer kurzen Beschreibung dargestellt. Dort sind auch weitere Programme erklärt, welche zu einem späteren Zeitpunkt in diesem Konzept verwendet werden.

Nach der Vorstellung der Stoffverteilung, in welcher die Inhalte grob auf die zur Verfügung stehenden Unterrichtsstunden aufgeteilt sind, werden die 4 Unterrichtseinheiten

- Objekt und Klasse,
- Beziehungen,
- Generalisierung und
- weitere Entwurfsmethoden

vorgestellt. Dabei wird nochmals ein nun etwas detaillierterer Überblick über die jeweilige Unterrichtseinheit gegeben, dann die Inhalte, Ziele und z.T. Methoden vorgestellt und diese begründet.

3.7.1 Stoffverteilung

Laut dem Lehrplan des informationstechnischen Gymnasiums in Baden-Württemberg [It01], in welchem dieses Konzept umgesetzt werden soll, umfasst die LPE 5 *Objektorientierte Analyse und Design* 45 Unterrichtsstunden. Hinzukommen noch ca. 15 Stunden zur Leistungsfeststellung und möglichen Vertiefung.

Der Stoffverteilungsplan (Tabelle 3) orientiert sich an der Vorgabe von 45 Stunden. Die restliche Zeit sollte für weitere Übungen, Klassenarbeiten und zum Eingehen auf spezifische Schülerprobleme genutzt werden. Die praktischen Teile werden zum Teil im Fach Computertechnik (CT) [Ct01] fortgeführt (siehe Spalte Bemerkung).

19 Programmiert wurde mit Java (jdk 1.4.2) und mit C++ (C++-Builder3 von Borland).

Unterrichtseinheit	Thema	Stunden	Bemerkung
Objekt und Klasse (9)	Programm analysieren	3	
	Programm anpassen	4	
	Konstruktor und Zugriffsrechte	2	inkl. Suchen eines neuen Algorithmus zum Gehen, Rest evtl. in CT
			Ein weiteres Beispiel ausgehend von einem Klassendiagramm in CT programmieren
Beziehungen (14)			
	Assoziationen und Kardinalitäten	4	Codierung in CT fortsetzten
	Rollen	2	Codierung evtl. In CT fortsetzen
	Aggregation und Komposition	2	
	Erste größere Analyse	2	
	Sequenz- und Kollaborationsdiagramm	4	Codierung in CT fortsetzten (8 Stunden)
Generalisierung (inkl. Polymorphie) (10)			
	Generalisierung	5	
	Klassen und Vererbungen finden (inkl. Polymorphie)	5	Codierung evtl. In CT fortsetzen
Weitere Entwurfsmethoden (10)			
	Zustandsdiagramme und der CVS-Pattern	6	Codierung in CT fortsetzten (4 Stunden)
	Anwendungsfälle (Use Cases)	2	
	Gesamtüberblick über OOAD	2	
Vertiefte Konzepte		0	Im ITG nicht relevant
Summe		43	

Tabelle 3. Der Stoffverteilungsplan des detaillierten Konzepts

Die letzte Unterrichtseinheit leitet bereits über zu einem anschließenden Projekt, wie es auch der Lehrplan in der LPE 7 [It01] mit 30 Stunden vorsieht. Dabei sollen die Schüler ihr Wissen anwenden und nach dem Vorgehen des objektorientierten Softwareentwurfs arbeiten. Parallel dazu sollen die Schüler die Methoden des Projektmanagements lernen, um das Projekt in der vorgegebenen Zeit zu meistern.

3.7.2 Objekt und Klasse

oder: Willi und Siggi wollen gehen lernen und sich begegnen

In dieser ersten Sequenz sollen die Grundbegriffe der Objektorientierung vermittelt werden. Eine Übersicht und die Lernziele finden sich in Tabelle 4, die konkrete Ausarbeitung im Anhang A.1.

Thema	**Lernziele**	**Bemerkungen**	**Stunden (IT)**
Programm analysieren	• Objekt(diagramm) aufstellen (C) • Klassen(diagramm) aufstellen (C) • Zusammenhang Code ↔ Diagramm erkennen! (B) • Attribute bestimmen (C) • Methoden bestimmen (C) • Referenzen verwenden	Code und Diagramme nur auf Papier (zum Lernen sollen die Schüler selbst zeichnen) Anstreichen der Attribute und Methoden im Code!	3
Programm anpassen	• Vertiefen obiger Lernziele (C) • Überladen anwenden (C) • Erste Erfahrung bei der Vorgehensweise beim Design machen (R)	Codieren der Programmerweiterung	4
Konstruktor und Zugriffsrechte	• Konstruktor konstruieren (C) • Datenkapselung (Information Hiding) anwenden (C)		2

Tabelle 4. Die Übersicht und die Lernziele der Einheit "Objekt und Klasse"[20]

Die Bedeutung der Begriffe *Objekt* und *Klasse* sollen klar und von den Schülern deutlich getrennt werden. Diese sind wichtig für ein Grundverständnis des objektorientierten Softwareentwurfs. Die Schüler sollen verstehen, dass Objekte real existieren, d.h. die Attribute konkrete Werte annehmen und im Speicher Raum beanspruchen, und dass Klassen als Baupläne für diese Objekte dienen. Daraus folgt dann, dass Objekte erzeugt werden müssen und auf diese zugegriffen werden kann. Dies erfolgt in Java über Referenzen in C++ über Variablen, Referenzen oder Zeiger. Hier muss je nach ausgewählter Sprache auch auf dies eingegangen werden. Das ist wichtig, um die Modelle später in ein Programm umsetzen zu können.

Mit der *Klasse* verknüpft sind zusätzlich die Begriffe *Attribut*, *Methode*, *Kapselung* (Information Hiding), die spezielle Methode des Konstruktors und Destruktors und das Überladen. Mit den Objekten sind die Verfahren, um auf Attribute und vor allem Methoden

20 Die Buchstaben in der Spalte *Lernziele* beziehen sich auf die Lehrzielmatrix in Tabelle 1. Lernziele ohne Referenzbuchstaben, sind in Bezug auf das gesamte Konzept von untergeordneter Wichtigkeit.

zugreifen zu können, verbunden. Dies alles soll hier vorgestellt und von den Schülern soweit verstanden werden, dass sie mit den Begrifflichkeiten umgehen und programmieren können.

Neben diesen wichtigen Fachbegriffen sollte sofort das Klassendiagramm eingeführt werden. Als weitere Hilfe zur Unterscheidung ist hier auch das Objektdiagramm wichtig. Der direkte Bezug zwischen Klassendiagramm und Code sollte anfangs immer wieder deutlich aufgezeigt werden, um langsam den Abstraktionsschritt zum Diagramm zu finden. Dies ist nötig, da mit den Diagrammen komplexere Sachverhalte übersichtlich darstellbar sind.

Im Unterrichtsablauf wird zuerst das Programm *Willis Welt* analysiert (siehe Software WillisWelt v1.0; Anhang D) .
Ausgehend von diesem Beispielprogramm (Oberfläche siehe Abbildung 16), von dem die Schüler recht einfach auf ihr reales Leben Bezug nehmen können, wird über die Kobolde *Willi* und *Siggi* der Begriff *Objekt* und über deren Eigenschaften wie Haut- oder Augenfarbe der Begriff *Attribut* eingeführt.
Über die Gemeinsamkeiten der beiden Objekte – beide haben die Eigenschaft Haut- bzw. Augenfarbe, allerdings mit unterschiedlicher Ausprägung – wird der Begriff *Klasse* und damit der Bauplan eingeführt. Der Unterschied zwischen dem Begriff *Objekt* und *Klasse* tritt zu Tage. Weiterhin kann dann der Begriff *Methode* über die Frage „Welche Aktionen kann der Kobold ausführen?“ erarbeitet werden. Über die Buttons *Willi zeigen*, *Siggi zeigen* und *verstecken* können die Kobolde Aktionen ausführen.

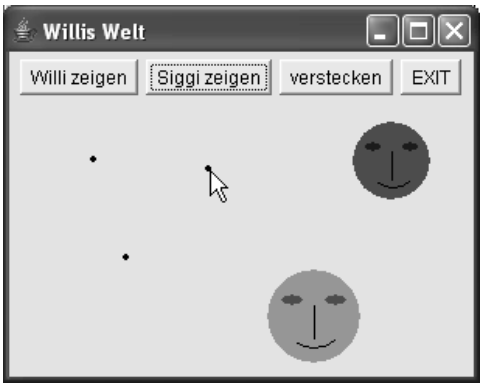


Abbildung 16. Das Fenster des Programms "Willis Welt“ (v1.0), mit Willi und Siggi

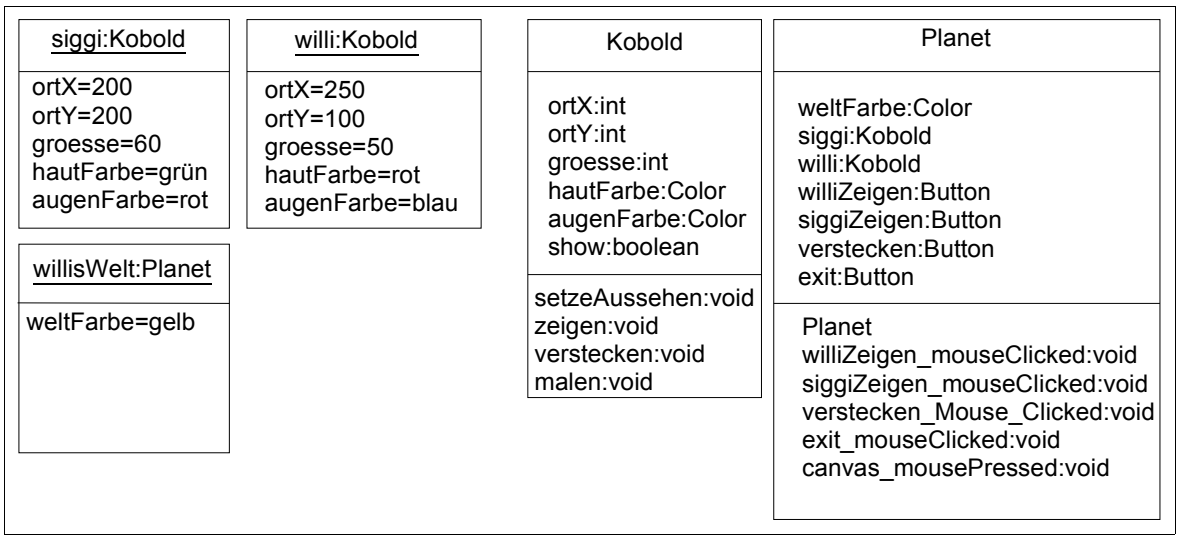


Abbildung 17. Die Objekte (links) und Klassen (rechts) des Programms "Willis Welt" (v1.0)

Diese Objekte und Klassen mit ihren Attributen und Methoden werden dabei parallel zum Lernfortschritt notiert (siehe Abbildung 17), wobei intuitiv das genormte Objekt- bzw. Klas-

sendiagramm entsteht, allerdings an dieser Stelle noch ohne Beziehungen untereinander, da dies erst in der nächsten Unterrichtseinheit Thema ist.

Da es sich um einführende Stunden handelt, wird als Unterrichtsmethode das Gespräch mit der Klasse vorgeschlagen, um zu verhindern, dass Unsicherheiten mit dem Thema entstehen. Zuerst sollen grundlegende Denkweisen vermittelt werden. Dazu gehört auch, dass die Schüler den Abstraktionsschritt vom Programmcode − der ihnen zum Teil aus der strukturierten Programmierung bekannt ist − zum Diagramm kennen lernen und einüben.

Nach Klärung der Begriffe und dem intuitiven Zeichnen des ersten Klassendiagramms wird die Verbindung zum Programmcode hergestellt, indem die Schüler die gefundenen Objekte, Klassen, Attribute und Methoden aus dem Diagramm im Programmcode suchen und markieren. Darin einbezogen ist die Klasse *Planet*, welche den Lebensbereich und damit die Steuerung der Kobolde darstellt. Damit ist den Schülern die Grundstruktur des gezeigten Programms bekannt.
Insgesamt lernen die Schüler dabei aus einem fertigen Beispiel und das mühsame Erstellen des ersten Programms entfällt. Der Einstieg in die recht komplexe Materie der Objektorientierung und die Verwendung der abstrakten Darstellung im Objekt- und Klassendiagramm gelingt spielerischer. Es besteht für den Schüler auch die Möglichkeit einfach an einem fertigen Programmgerüst Änderungen vorzunehmen und die Auswirkungen zu erkunden.

Es kann nun das erste Mal eine strukturierte Erweiterung des Programms erfolgen, indem die Schüler die Aufgabe erhalten, dass die Kobolde nun gehen sollen. Durch einen *Klick* auf die Oberfläche des Programms sollen sich die Kobolde ein kleines Stückchen auf diesen Punkt zu bewegen.
Wichtig ist dabei, im Erweiterungsprozess (Entwurfsprozess) strukturiert über die neuen nötigen Fähigkeiten der Klassen zu sprechen, diese im Klassendiagramm einzutragen und erst dann den Programmcode anzupassen. Dadurch ergibt sich die neue Methode *gehen* und das neue Attribut *schrittweite*.
Vor der Codierung muss allerdings noch auf die Problematik der Ereignissteuerung und deren Verarbeitung im Programm eingegangen werden.
Die Software ist dabei bereits so weit vorbereitet, dass ähnliche Abläufe bereits enthalten sind. So wird in der Version 1.0 (WillisWelt v1.0) bereits ein kleiner schwarzer Fleck auf die Stellen gezeichnet, auf die der Anwender in *Willis Welt* geklickt hat (siehe auch Abbildung 16). Die Schüler müssen an dieser Stelle im Programmcode die neue Methode *gehen* der Objekte *willi* bzw. *siggi* der Klasse *Kobold* einfügen.
Die Schüler sollen das Programm jeweils selbständig entsprechend erweitern und die Auswirkungen testen. Dies kann je nach Klasse in Einzel-, Partner- oder Gruppenarbeit erfolgen.

Ähnlich wird das Überladen eingeführt. Dabei unterscheiden sich zwei oder mehrere Methoden einer Klasse nicht in ihrem Namen, sondern nur in der Parameterliste, welche übergeben wird. Hier soll eine weitere Möglichkeit des Gehens verwendet werden, indem die Geh-Richtung durch Polarkoordinaten festgelegt wird.

Über unveränderliche Eigenschaften der Kobolde (z.B. der Augenfarbe), welche bei der Geburt festgelegt werden sollen, dann aber nicht mehr geändert werden dürfen, wird der Konstruktor zur Initialisierung der Eigenschaften und die Zugriffsrechte[21] eingeführt.

Um die Inhalte zu vertiefen und um die Umsetzung des Gelernten auf andere Bereiche zu üben sollte das Gelernte an den entsprechenden Stellen an anderen Problemstellungen vertieft werden. So kann z.B. an von den Schülern vorgeschlagenen Beispielen der Unterschied zwischen Klasse und Objekt geübt werden. Als weiteres kleines Projekt können Autos durch Klicken über den Bildschirm „fahren“.

3.7.3 Beziehungen

oder: Die Kobolde möchten einen König haben

Zuerst sollen hier weitere Grundbegriffe geklärt werden. Eine Übersicht und die Lernziele der Einheit *Beziehungen* finden sich in Tabelle 5, die konkrete Ausarbeitung im Anhang A.2.

Untereinheit	**Lernziele**	**Bemerkungen**	**Stunden (IT)**
Assoziationen und Kardinalitäten	• Beziehungen zwischen Klassen erkennen (E) und bestimmen (F) • Darstellungen in UML anfertigen (Y) • Bezeichnungen von Assoziationen finden (F) • Kardinalitäten bestimmen (F) • Beziehungen im Programmcode anwenden (F) • Klassendiagramme teilweise entwickeln (H,S) • Domain Models (OOA) entwerfen (H,S)	Codierung evtl. in CT fortsetzten	4
Rollen	• Rollen und Rollennamen anwenden (F,Y) • Erweiterung eines Klassendesigns (Domain Model) mit Rollen (F)	Codierung evtl. in CT fortsetzen	2
Aggregation und Komposition	• Komposition und Aggregation abgrenzen und anwenden (F)		2
Erste größere Analyse	• größere Domain Models entwerfen (OOA) (H,S) • Vorgehen beim Design analysieren und anwenden (G) • Vor-/Nachteile der Modelle		2

21 Nur die Zugriffsrechte *public* und *private* und nicht *protected*, da die Generalisierung den Schülern noch nicht bekannt ist.

Untereinheit	Lernziele	Bemerkungen	Stunden (IT)
	einschätzen (I) • Problem des Designs abhängig von der Problemstellung erkennen (I)		
Sequenz- und Kollaborationsdiagramm	• Sequenz- und Kollaborationsdiagramm in UML anfertigen (F,Y) • Sequenz- und Kollaborationsdiagramm beim Designprozess verwenden (H,S) • Designrichtlinien für gutes Design kennen und anwenden (H,S) und gefundene Lösungen damit bewerten (I) • größeres Projekt entwickeln (H,S)), unter Bewertung von Zwischenlösungen(I)	Codierung evtl. in CT fortsetzten (8 Stunden)	4

Tabelle 5. Die Übersicht und die Lernziele der Einheit "Beziehungen"[22]

Bisher konnte man eigentlich nicht von einem Klassendiagramm sprechen, da nur einzelne Klassen betrachtet wurden. Nun kommen mehrere Klassen ins Spiel und die Beziehungen untereinander sollen geklärt werden. Dabei ist die Frage der Kardinalitäten, der Rollen und der besonderen Formen der Assoziation als Komposition und Aggregation von Bedeutung.

Über die Leitfrage „Welche Verbindung besteht zwischen der Klasse *Kobold* und der Klasse *Planet*?“ wird dieses Thema aufgegriffen und in das Klassendiagramm übernommen. Eigentlich ist dieser Schritt nur die Einführung einer Notation im Klassendiagramm. Allerdings legt er den Fokus auch auf diese für den Entwurf wichtige Problematik. Es wird hier der Weg vom Speziellen zum Allgemeinen gewählt, denn im Programmcode von *Willis Welt* ist die Beziehung zwischen der Klasse *Kobold* und *Planet* bereits vorhanden.
Die Kardinalität ist bereits beim Analysemodell, also bei der Beschreibung des Problems und nicht erst bei dessen Lösung wichtig. Dadurch wird die Realität genauer beschrieben, da eine Aussage getroffen wird, dass z.B. bis zu 4 Kobolde (Kardinalität 0..4) auf dem Planet leben und jeder Kobold auf genau einem Planet (Kardinalität 1) lebt (siehe Abbildung 18, ohne Klasse *Krone*).

Unterstützt und vertieft wird dies über ein Arbeitsblatt (siehe Anhang B.1) mit drei Aufgaben, welche jedoch getrennt zu bearbeiten sind. In der ersten Aufgabe sind zu mehreren Beispielen die Kardinalitäten gefragt. Nach Bearbeitung durch die Schüler und der Besprechung im Unterrichtsgespräch sollen die Schüler im bestehenden Programmcode nach der Realisierung der Beziehungen suchen. Bei der Besprechung dieser zweiten Aufgabe wird die Navigierbarkeit eingeführt. Diese gibt an, von welcher zu welcher Klasse die Beziehung besteht. Die Beziehung erhält dadurch eine Richtung. Allerdings ist diese Richtung meist

22 Siehe Fußnote auf Seite 45.

erst in der Codierungsphase wichtig, da sie festlegt, auf welcher Seite das Attribut für die Assoziation abgelegt werden soll. Sie bereitet dadurch bereits die dritte Aufgabe vor.

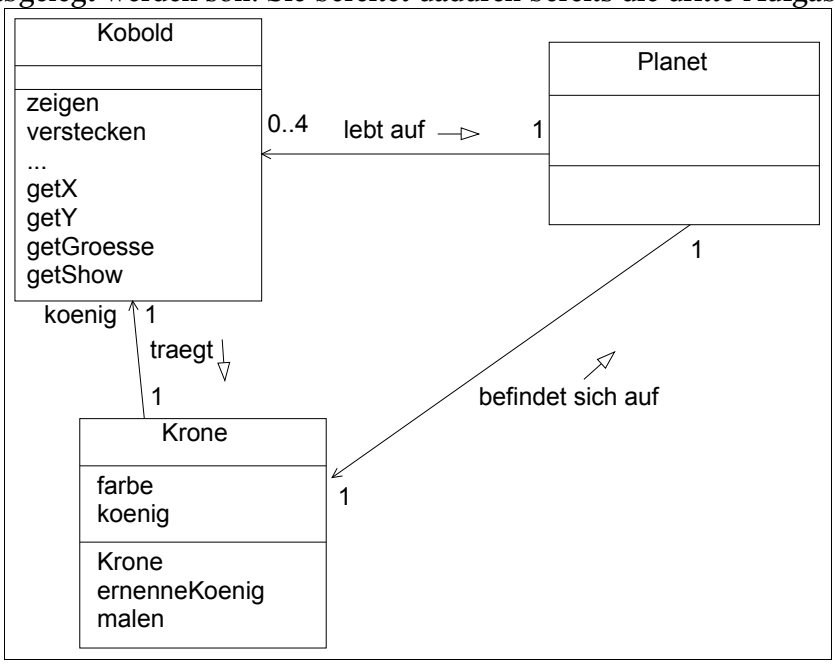


Abbildung 18. Das Klassendiagramm mit der Rolle „Koenig“ und der Klasse „Krone“

Diese dritte Aufgabe dreht sich nun um die Realisierung größerer Kardinalitäten im Programmcode (Kardinalitäten größer 1). Dies ist eine Umsetzung des gefundenen Klassendiagramms in das Programm, also der strukturierte Weg für die Umsetzung zur Lösung. Für kleine endliche Kardinaliäten bietet sich dazu ein statisches Array als Attribut an[23]. Nach Besprechung dieser Möglichkeit sollte dies von den Schülern in Einzel-, Partner- oder Gruppenarbeit (je nach Randbedingungen) im Programmcode umgesetzt werden.

Nun sollten die Schüler soweit mit den Beziehungen vertraut sein, dass zum ersten Mal ein Modell ausgehend von einer Problemstellung gesucht werden kann (objektorientierte Analyse). Über eine kleine Aufgabe, in der – um die Arbeit zu erleichtern – die Klassen bereits vorgegeben sind, sollen die Beziehungen, Kardinalitäten und Attribute gefunden werden. Um das strukturierte Arbeiten zu lernen, gibt ein Arbeitsblatt (siehe Anhang B.2) die Schritte vor. Anschließend sollten die von den Schülern gefundenen Lösungen besprochen werden.
Danach wird dieses allgemeine Vorgehen über ein schwierigeres Problem (siehe Anhang B.3 [Br01]) nochmals geübt.

Im nächsten Schritt sollen Rollen eingeführt werden. Im hier verwendeten Programm soll ein Kobold die Rolle des Königs übernehmen. Dies wäre allerdings auch über eine Vererbung lösbar, bei der der König eine Spezialisierung eines Kobolds ist. Der Nachteil ist jedoch, dass damit der König für die gesamte Laufzeit des Programms festliegt, da nur er ein Objekt der einzig passenden Klasse ist. Bei der Lösung über Rollen kann der König

23 Die Realisierung größerer Kardinalitäten bzw. Kardinalitäten mit unbestimmter Größe wird später besprochen.

während der Laufzeit des Programms geändert, bzw. neu *gewählt* werden. Die Lösung ist flexibler. Gut zu erkennen ist, dass das Konzept der Rolle in manchen Situationen eine Alternative zur Spezialisierung darstellt.
Nach Einführung des Begriffs am Beispiel von *Willis Wel*t werden weitere Beispiele gesucht und besprochen. Gut eignet sich dabei, diesen Begriff auf unser Leben zu beziehen, so kann eine Person je nach Umgebung mehrere Rollen spielen: Sohn, Schüler oder Mitspieler im Verein.

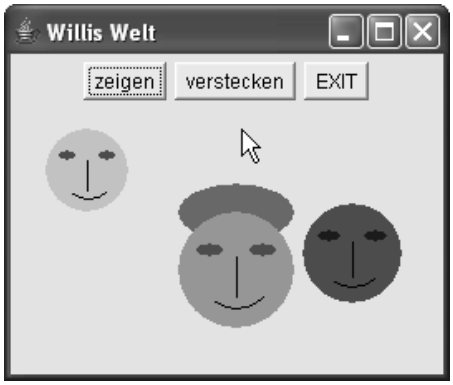


Abbildung 19. "Willis Welt" (v1.9)mit 3 Kobolden, der Krone und dem König

Danach wird wiederum das strukturierte Vorgehen beim Softwareentwurf geübt, indem die Attribute und Methoden für die Klassen nach Erweiterung von *Willis Welt* gesucht werden. Dabei kann man an dieser Stelle bereits auf die Verantwortlichkeiten der einzelnen Klassen eingehen und entsprechend die neuen Attribute und Methoden zuweisen. Dieses Thema wird jedoch am Ende dieser Unterrichtseinheit noch genauer angesprochen.
Erst nach der Erweiterung des Klassendiagramms (siehe Abbildung 18) wird alles von den Schülern im Programm umgesetzt (siehe Abbildung 19). Hierbei sind evtl. Softwaretools zum Zeichnen der Diagramme und zur Umsetzung dieser in ein Programmgerüst hilfreich (z.B. Together [To02]).

Als Einschub kann an dieser Stelle das Thema *Aggregation und Komposition* kommen. Die Unterscheidung zwischen einer Aggregation und einer Komposition ist für die Analyse und das Design meist nicht so bedeutend. Allerdings wird hierbei geschult, die Probleme etwas genauer zu beleuchten. So stellt eine Aggregation eigentlich nur eine Teil-Ganzes-Beziehung (z.B. eine Person als Teil eines Freundeskreises) dar. Eine Komposition geht einen wichtigen Schritt weiter, indem eine existentielle Abhängigkeit zwischen dem Teil und dem Ganzen besteht. So stirbt das Teil (die Finger), wenn das Ganze (die Hand) stirbt.
Eine kleine Übung (siehe Anhang B.4) zu dieser Thematik von T. Brinda [Br01] schließt diesen Einschub ab.

Gegen Ende dieser Einheit sind dann die Grundbegriffe so weit geklärt, das man auf das Ziel der Problemlösefähigkeit zusteuern kann. Dabei handelt es sich um einen kreativen Prozess, welcher aber durch die Randbedingungen der Programmiersprache begrenzt wird. Diese Randbedingungen sind den Schülern an dieser Stelle soweit bekannt, dass sie diese Herausforderung annehmen können. Auch in der gesamten Technik stößt man auf diese Problematik: Der kreative Prozess beim Design eines mechanischen oder elektrischen Bauteils unterliegt dabei den physikalischen Gegebenheiten. Die Möglichkeiten sind dadurch begrenzt, manche Probleme sind unlösbar oder nur näherungsweise lösbar.
Ein Problem genau zu erfassen ist ein wichtiger Schritt, um das Problem zu lösen. Dies erfolgt hier durch das Erstellen eines Analysemodells (OOA, objektorientierte Analyse). Deshalb wird auch dieser Begriff eingeführt. Im Vordergrund steht zuerst, das Problem genau zu erfassen und in ein Modell umzusetzen. Nicht das WIE sondern das WAS ist gefragt. Viele Softwareprojekte scheitern deshalb, weil nicht verstanden wurde, welche

Anforderungen gestellt wurden [La02, S. 42]. Wichtig ist, diesen kreativen Prozess zusammen mit den Schülern zu analysieren und zu strukturieren, so dass sie für weitere Entwurfsprozesse gut gewappnet sind.
Über ein Arbeitsblatt (siehe Anhang B.5) mit einem kurzen Dialog zwischen Anwender und Entwickler über das Spiel TicTacToe, sollen die Schüler den Schritt zur Problemlösung gehen. Als Hilfe sind dabei die einzelnen Schritte beim Entwurf angegeben, an Hand derer sie zur Lösung, dem Klassendiagramm, gelangen sollen. Nach dem Entwurf in Einzel-, Partner- oder Gruppenarbeit und deren Besprechung erfolgt in einem Unterrichtsgespräch die Zusammenfassung der Entwurfsschritte. Dabei wird an diesem Beispiel der allgemeine Verlauf beim Entwurf verdeutlicht und dann allgemein formuliert (siehe Anhang A.2.5, Abbildung 55). Hier wird der induktive Weg vom speziellen Beispiel zur allgemeinen Vorgehensweise gewählt.

Als ein weiterer Einschub eignet sich an dieser Stelle eine kleine Aufgabe von G. Booch ([Bc94], siehe Anhang B.6), in der folgende Problematik angesprochen wird: Um eine Lösung korrekt zu finden, müssen möglichst viele Informationen gesammelt werden, sollte also das Problem gut abgesteckt werden.

Mit den bisher vorgestellten Verfahrensweisen findet man oft nicht alle wichtigen Methoden für die Klassen. Eine große Hilfe ist dabei das Sequenzdiagramm. Deswegen wird es anschließend im Zusammenhang mit der Problematik des Findens aller wichtigen Methoden eingeführt, um dessen Bedeutung und Verwendung darzustellen.
Am stark vereinfachten Beispiel des Abhebens an einer Bank wird das Sequenzdiagramm vorgestellt und seine Bedeutung erkennbar (siehe Anhang A.2.6, Abbildung 56). Es stellt den zeitlichen Ablauf der Kommunikation der einzelnen Objekte übersichtlich dar und hilft, indem man darin den Ablauf Schritt für Schritt notiert, beim Finden der Methoden. Im Gegensatz zum statischen Klassendiagramm sind mit dem Sequenzdiagramm nun auch dynamische Abläufe sichtbar.
Anschließend werden die Erkenntnisse am Beispiel des Schaltens bei einem Auto (siehe Anhang B.7) eingeübt und vertieft. Da voraussichtlich unterschiedliche Lösungen gefunden werden, kann an dieser Stelle auf die Bewertung dieser eingegangen werden. Dies ist sehr anspruchsvoll, da hierbei die nach Bloom höchste Taxonomiestufe (Evaluation) von den Schülern gefordert wird (siehe Kapitel 3.5). Als Hilfe werden ihnen die Designrichtlinien von Larman [La02] angeboten, welche Regeln für ein gutes Design darstellen (siehe Anhang A.2.6, Abbildung 57). Um eine gute Modularisierung in Form von Klassen zu erhalten, sollte jede Klasse eine konkrete Aufgabe übernehmen und möglichst unabhängig von anderen Klassen agieren können. Aus der zugewiesenen Aufgabe folgt dann auch direkt, welche Informationen (d.h. Attribute) dort abgelegt sein sollten.
Eine Schwierigkeit ist dabei, dass sich diese Richtlinien zum Teil widersprechen, so dass von den Schülern eine Abwägung verlangt wird. Dieser Prozess ist jedoch als Lernziel äußerst wichtig, da es auch sonst oft vorkommt, dass sich Ziele widersprechen und abgewogen werden müssen. An dieser Stelle können die Schüler damit Erfahrungen sammeln.

Anschließend kann das oben gefundene Analysemodell des TicTacToe-Spiels mit Hilfe von Sequenzdiagrammen um Methoden erweitert und dann codiert werden. Allerdings ist dies

sehr zeitaufwendig, bietet aber eine hohe Motivation und einen großen Erkenntnisgewinn für die Schüler.
Dabei wird bereits der Weg zum Design (OOD, objektorientiertes Design) gegangen. Das WIE steht im Vordergrund, also WIE das Problem zu lösen ist und nicht mehr WAS das Problem ist.

3.7.4 Generalisierung

oder: Die Kobolde wünschen sich die Unterteilung in Mann und Frau

Bisher wurde auf das durch die objektorientierte Programmierung neue Konzept der Vererbung nicht eingegangen. Im Softwareentwurf spricht man anstatt von *Vererbung* eher von *Generalisierung*, da generalisierte Eigenschaften und Methoden verwendet werden. Diese mächtige Errungenschaft vereinfacht einerseits den Entwurf, da man auf fertige Konstrukte recht einfach zurückgreifen kann (Codewiederverwendung), andererseits hilft er auch bei der Strukturierung des Problems und bietet so eine weitere Hilfe bei der Findung der Problemlösung. Auch diese Möglichkeit entspricht wiederum dem menschlichen Denken. So ist die Generalisierung (bzw. Spezialisierung) z.B. in der Biologie bei der Strukturierung der Pflanzen- und Tierwelt ein gängiges Verfahren (siehe Abbildung 60, Anhang A.3.2).
Eine Übersicht und die Lernziele finden sich in Tabelle 6, die detaillierte Ausarbeitung im Anhang A.3.

Thema	**Lernziele**	**Bemerkungen**	**Stunden (IT)**
Generalisierung	• Generalisierungen konstruieren (O) • Überschreiben von Methoden anwenden (M) • Vererbung programmieren • Zugriffsrecht *protected* anwenden (M) • abstrakte Klassen und Methoden konstruieren (O) • Klassendiagramme mit Generalisierung analysieren (P)		5
Klassen und Vererbungen finden (inkl. Polymorphie)	• Polymorphie konstruieren (O) • dynamische Arrays anwenden • Klassendiagramme mit Generalisierung entwerfen und bewerten (P,S)		5

Tabelle 6. Die Übersicht und die Lernziele der Einheit "Generalisierung"[24]

Methodisch wird in einem Unterrichtsgespräch das den Schülern bereits bekannte Programm mit Willi und Siggi (ausgehend von der Version v1.9) wiederum erweitert. Dies hat

24 Siehe Fußnote auf Seite 45.

den Vorteil, dass eine größere Einarbeitungszeit entfällt. Es sollen nun zwei Arten von Kobolden erzeugt werden: Männer und Frauen (siehe Abbildung 20).

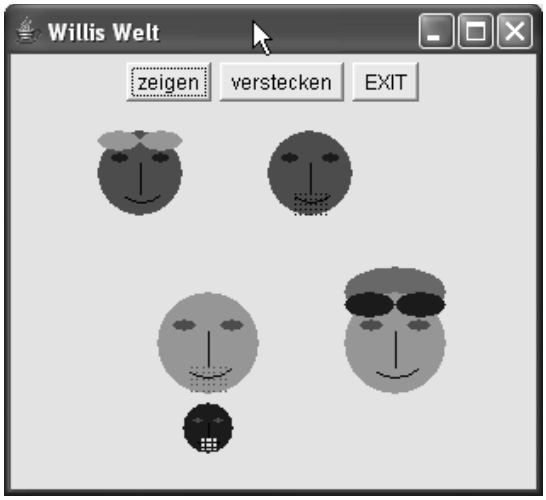


Abbildung 20. "Willis Welt" (v1.11) mit Männern (Bart) und Frauen (Haarschleife) und der Königin

Die Gemeinsamkeiten und Unterschiede der beiden Klassen lassen sich hier gut herausarbeiten und ausgehend vom oben Besprochenen ergibt sich die Lösung des Problems mit Hilfe der Generalisierung.
Bei diesem Vorgehen werden zuerst zwei getrennte Klassen, *KoboldMann* und *Kobold-Frau* entworfen, im nächsten Schritt werden die Gemeinsamkeiten gesucht (gemeinsame Attribute und Methoden) und daraus die Superklasse gebildet. Zum Schluss wird diese Lösung sprachlich über das Satzkonstrukt *ist ein* überprüft (Ein KoboldMann *ist ein* Kobold). Zusätzlich wird überprüft, ob alle Attribute und Methoden der Superklasse auch für die Subklasse gelten. Ist dies nicht der Fall, so ist die Generalisierung zumindest in Frage zu stellen. Die Unterschiede der Subklassen sollten sich auch nur in den Subklassen zeigen.

An weiteren Beispielen, welche von den Schülern gesucht werden sollen, wird die Generalisierung anschließend vertieft und die Frage aufgeworfen: „Wann ist Generalisierung sinnvoll?“ Als Test werden wieder die obigen beiden Regeln verwendet:

- Lassen sich die Klassen über das Satzkonstrukt *ist ein* verbinden? (z.B. Ein Auto *ist ein* Landfahrzeug)
- Sind alle Attribute und Methoden der Oberklasse auch für die Unterklasse sinnvoll?

Als kleiner Exkurs ist das Beispiel aus der Biologie gedacht (siehe Abbildung 60, Anhang A.3.2). Es zeigt, dass das Konzept der Generalisierung (Vererbung) auch in anderen wissenschaftlichen Disziplinen angewandt wird und dient als weiterer Anker für das Gelernte.

Als nächster Schritt wird nun über ein Arbeitsblatt (siehe Anhang B.9) das Finden einer Generalisierung verlangt. Über die Reihenfolge der Aufgaben wird der Schüler wieder zum strukturierten Arbeiten angeregt. Dabei steht zuerst nicht die Generalisierung im Vordergrund, sondern die Objekte und Klassen. Erst danach wird über Gemeinsamkeiten der Klassen die Generalisierungsstruktur gesucht. Das allgemeine Vorgehen wird anschließend über das Vorgehen an diesem Beispiel vergegenwärtigt.

Nun, nachdem auf der Ebene der Diagramme die Lösung gefunden und verstanden wurde, kann das Programm *Willis Welt* ausgehend von v1.9 in Einzel-, Partner- oder Gruppenarbeit codiert werden. Dazu muss jedoch erst noch die Syntax der Vererbung vorgestellt werden. Steht ein entsprechendes Case-Tool zur Verfügung (z.B. Together [To02]), dann kann dieser Schritt hierüber erfolgen. Die Schüler zeichnen damit die Vererbung und erhalten auto-

matisch die Syntax in der Programmiersprache. Das beim Compilieren auftretende Problem des Zugriffsrechts *private* wird über das neue Zugriffsrecht *protected* gelöst. Dieses Zugriffsrecht gewährt allen Kindklassen den Zugriff auf Attribute bzw. Methoden, nicht jedoch den anderen Klassen.
Hier sollte auch auf weitere Probleme bei der Umsetzung der Klassendiagramme in Programmcode eingegangen werden: Aufruf des Konstruktors der Superklasse, Verwendung von Methoden in der Superklasse, welche in der Subklasse überschrieben wurden etc. Diese sind jedoch sprachabhängig.
An dieser Stelle sollte auch erwähnt werden, dass die Vererbung die Pflege des Programmcodes vereinfacht: Änderungen in der Superklasse wirken sich automatisch auf alle Subklassen aus.

Es soll nun verhindert werden, dass Objekte der Klasse *Kobold* erzeugt werden können, da nur noch die Klassen *KoboldMann* und *KoboldFrau* verwendet werden sollen. Dies führt direkt zum Begriff der *abstrakten Klasse*. Auch die Methode *malen()* benötigt in der Klasse *Kobold* nun keine Implementierung mehr, sie wird zur abstrakten Methode, also zur Methode ohne Implementierung. Dies erzwingt automatisch eine *abstrakte Klasse*. Um Objekte von Unterklassen zu erzeugen, müssen diese alle abstrakten Methoden der Oberklasse überschreiben, d.h. in der Unterklasse wird eine Methode mit identischem Namen und Parameterliste verwendet und auch implementiert.
Eventuell kann man hier nochmals auf das Benennungssystem der Pflanzen- und Tierwelt in der Biologie zurückgreifen (siehe Abbildung 60, Anhang A.3.2). Auch dort sind die Äste des Benennungsbaums abstrakte Klassen, da es diese *Tiere* real nicht gibt. Nur die Blätter dieses Baums sind nicht abstrakt, also real existierende Klassen für Pflanzen bzw. Tiere.
Für die Umsetzung in C++ muss in einem Unterrichtsgespräch noch der Begriff *virtuelle Klasse* eingeführt werden. In Java ist dies nicht nötig, da dort alle Klassen virtuell sind.

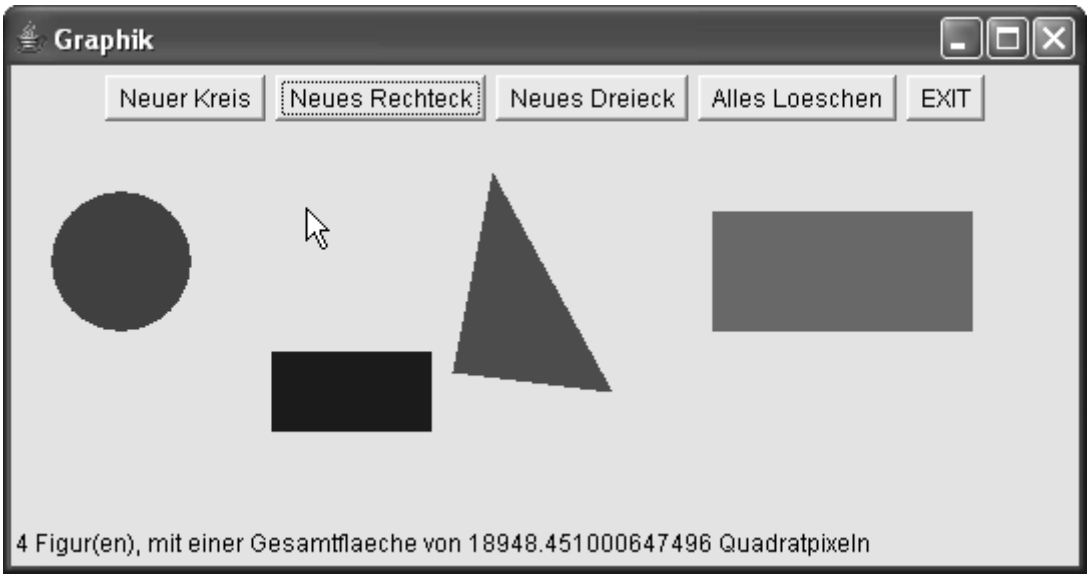


Abbildung 21. Die Oberfläche zum Programm „Graphik“ (v1.5)

Das Thema abstrakte Methoden und Klassen bereitet bereits die Polymorphie vor. Zuerst wird jedoch das bisher Gelernte zum Bereich Generalisierung in einem kleinen Projekt als

Analyse-Entwurf umgesetzt. Dort sollen 2-dimensionale Figuren gezeichnet und deren Flächeninhalt berechnet werden (siehe Abbildung 21).
Beim Entwurf des Domain Models (Analyse-Modell) sollte man bei den Schülern auf das bisher vermittelte strukturierte Vorgehen achten und das Ergebnis in Bezug auf die Design-Richtlinien (siehe Kapitel 3.7.3) und auf die Regeln bei der Generalisierung überprüfen, sodass die Schüler dieses Vorgehen verinnerlichen.

Der Übergang zum Design-Modell führt dann über ein Unterrichtsgespräch die Polymorphie[25] (dt.: Vielgestaltigkeit) ein, bei der erst zur Laufzeit entschieden werden kann, welche Methode verwendet wird. Sie erspart größere Auswahlabfragen (switch cases), bietet so eine große Flexibilität im Code und wird erst durch das Konzept der Vererbung möglich. Damit lassen sich Objekte unterschiedlicher Subklassen, welche jedoch alle dieselbe Superklasse besitzen, in dasselbe Feld speichern.
Damit ist erst eine Beziehung einer Klasse zu einer Superklasse mit mehreren Subklassen möglich. Das Beziehungs-Attribut erhält den Datentyp der Superklasse, das abgelegte Objekte ist jedoch evtl. vom Datentyp der Subklasse. Ein Methodenaufruf führt automatisch zur korrekten Methode der Subklasse, welche eine Methode der Superklasse überschrieben hat. Besonders vorteilhaft ist dieses Verhalten, wenn das Beziehungs-Attribut ein Feld ist, d.h. eine Kardinalität größer als 1 verwendet wird.
Dieses Verhalten kann gut mit dem Projekt zum Zeichnen 2-dimensionaler Figuren dargestellt werden (Klassendiagramm siehe Abbildung 13, Kapitel 3.6.3). Eine weitere Möglichkeit besteht darin, dass im Beispiel von *Willis Welt* die Klassen *KoboldMann* und *Kobold-Frau* in einem gemeinsamen Feld abgelegt werden.
Nebenbei ergibt sich die Problematik, die Kardinalität *viele* im Programmcode umzusetzen. Dies gelingt mit Hilfe von Containern (dynamisches Feld), welche z.B. in Java oder in C++[26] in den Standard-Bibliotheken vorhanden sind. Diese sind im Endeffekt in Klassen gekapselte Listen und können ohne deren Kenntnis verwendet werden. Die Verwendung der Container wird über ein Arbeitsblatt vermittelt (siehe Anhang B.11) und wird je nach Projektfortschritt an die Gruppen verteilt.

Bei der Umsetzung des Designs in Programmcode sollte darauf geachtet werden, dass gut dokumentiert wird und die einzelnen Subklassen getrennt getestet werden. Eine Musterlösung für dieses Projekt ist das Programm *Graphik* (v1.1 – 1.5, Anhang D). Das Klassendiagramm zur Version v1.3 findet sich zusammen mit dem Sequenzdiagramm zum Neuerstellen eines Graphik-Objekts im Anhang B.10.

3.7.5 Weitere Entwurfsmethoden

Als letzte Einheit des Konzepts (Übersicht siehe Tabelle 7; detaillierte Ausarbeitung siehe Anhang A.4) liefert die Automatentheorie mit dem Zustandsdiagramm noch eine wichtige

25 Mit Polymorphie ist hier nicht das Konzept des Überladens (siehe Kapitel *Objekt und Klasse*) gemeint, wie es teilweise in der Literatur zu finden ist. Unter Polymorphie wird hier verstanden, dass gleichnamige Methoden mit identischen Parameterlisten verwendet werden, welche in unterschiedlichen Subklassen unterschiedlich implementiert sind.

26 Containerklassen z.B. in Java in der Klasse java.util.Vector, in C++ in der Klasse vector der Standard Template Library [ISO98] bzw. bei älteren Entwicklungsumgebungen in entsprechenden Bibliotheken

Hilfe beim Entwurf. Der Schwerpunkt liegt dabei weniger auf der Automatentheorie, sondern wieder beim Weg vom Problem zur Lösung.
Umgesetzt wird dies an dem praktischen Beispiel eines Zahlenschlosses, bei dem 3 Zahlen in der richtigen Reihenfolge eingegeben werden müssen, um das Schloss zu öffnen. Dabei werden die inneren Zustände unmittelbar klar: der Automat muss wissen, dass z.B. bereits zwei richtige Zahlen eingegeben wurden und er auf die richtige dritte Zahl nun mit Öffnen des Schlosses reagieren muss.
Über die verbale Beschreibung des Verhaltens des Zahlenschloss-Automaten gelangt man zur Erkenntnis, dass der Automat sich intern merken muss, wie viele richtige Zahlen in der Zahlenfolge bereits eingetippt wurden.

Thema	**Lernziele**	**Bemerkungen**	**Stunden (IT)**
Zustandsdiagramme und das CVS-Pattern	• einen Automaten mit seinem äußeren und inneren Verhalten erklären • einen einfachen Automaten mit Hilfe des Zustandsdiagramms entwerfen • den Automaten programmieren • die CVS-Architektur beschreiben und anwenden (S)	Maier-Suchmaschine in CT entwickeln und programmieren.	6
Anwendungsfälle (Use Case)	• Anwendungsfälle suchen und beschreiben (W)		2
Überblick über den ges. Entwurfsprozess (Unified Process)	• den allgemeinen Ablauf beim objektorientierten Softwareentwurf verstehen (R,U)	Die Anwendung des Entwurfablaufs erfolgt im anschließenden Projekt.	2

Tabelle 7. Die Übersicht und die Lernziele der Einheit „Weitere Entwurfsmethoden“[27]

Im nächsten Schritt erfolgt die Umsetzung der verbalen Beschreibung in ein Diagramm, dem Zustandsdiagramm (siehe Abbildung 68, Anhang A.4.2). Nachdem die Anzahl der Zustände ermittelt wurde, wird zuerst der Weg von Zustand zu Zustand gesucht – unter der Annahme, dass die richtigen Zahlen hintereinander eingegeben wurden. Anschließend kümmert man sich um die Wege, bei welchen Störungen, wie z.B. eine falsche Zahl, eingegeben wurden.

Nach Übungen zur Vertiefung (siehe Arbeitsblatt 11, Anhang B.12), erfolgt danach die Betrachtung des Automaten als Black-Box. Dadurch ist es den Lernenden einfacher möglich, diese Betrachtungsweise anzunehmen. Daraus folgt dann sofort das Klassendiagramm (siehe Abbildung 69, Anhang A.4.2). Dieses entspricht im Prinzip der Black-Box-Darstellung, da nur die Schnittstellen (hier: Methoden) der Maschine gesucht werden. Auf diese

27 Siehe Fußnote auf Seite 45.

Weise ergibt sich der Ansatz, den Automaten in einer Klasse abzulegen. Die Designrichtlinie *Informationsexperte* (siehe Kapitel 3.4), an jeder Klasse eine gezielte Aufgabe zu vergeben, ist hiermit erfüllt.

Aber auch die Konstruktion einer Zustandsmaschine bis zu deren Bau (siehe Programm *Zahlenschloss*, Anhang D) soll erarbeitet werden. Es soll jedoch deutlich werden, dass der Übergang vom Zustandsdiagramm zum Programmcode durchaus von einer Maschine übernommen werden kann und der Großteil der geistigen Leistung sich im Entwurf des Zustandsdiagramms, also des eigentlichen Automaten, befindet.
Die Unterscheidung zwischen Mealy-Automat, bei dem das auslösende Ereignis X und der innere Zustand Z über die Ausgangsrelation λ den Ausgangswert Y beeinflussen, und dem Moore-Automat, bei dem nur der innere Zustand Z über die Ausgangsrelation λ den Ausgangswert Y beeinflusst, spielt in der softwaretechnischen Realisierung der Automaten eine kleinere Rolle[28], sodass dieses Thema nicht angesprochen werden muss. Sie spielt nur in Form der graphischen Notation eine geringe Rolle, indem beim Mealy-Automaten der Ausgangswert Y am Übergang, beim Moore-Automat im Zustand angegeben wird. Beide Möglichkeiten sollen angesprochen werden, ohne jedoch unbedingt auf die begriffliche Unterscheidung der beiden eingehen zu müssen.
Zur softwaretechnischen Realisierung eines Automaten existieren zwei Möglichkeiten:

- Über Auswahlabfragen bzw. *switch-case*-Anweisungen bzw.
- über Tabellen.

Der tabellengesteuerte Automat ist einfacher zu realisieren und zu verstehen, so dass er hier ausgewählt wurde.
Der zunächst recht abstrakt anmutende Ansatz mit der Übergangsrelation δ und der Ausgangsrelation λ wird dann praktisch umgesetzt, indem für jeden Zustand und jedes mögliche Ereignis das Ergebnis dargestellt wird. Dies lässt sich am einfachsten über Tabellen darstellen, der Schritt von der abstrakten Relation zur anschaulicheren Tabelle wird vollzogen. Daraus ergibt sich automatisch der Ansatz, die Tabellen über die den Schülern bereits bekannten 2-dimensionalen Arrays zu lösen. Dazu muss nur noch jedem Zustand und jedem Ereignis eine Zahl zugeordnet werden.
Über die rein sprachliche Beschreibung des Ablaufs für die Zustandsmaschine können die Schüler nun selbständig den Algorithmus in Form eines Struktogramms erstellen (siehe Abbildung 71, Anhang A.4.2).
Die Automatisierbarkeit des Übergangs vom Zustandsdiagramm zum Programmcode kann verdeutlicht werden, indem der universelle Algorithmus für den tabellengesteuerten Automaten erarbeitet wird. Dabei wird klar, dass die eigentliche kreative Arbeitsleistung im Finden des Zustandsdiagramms besteht, die folgenden Schritte sind dann einem Kochrezept ähnlich umsetzbar.

Ein weitere Aufgabe dieser Einheit ist die Vorstellung der CVS-Architektur (Control, View, Storage, siehe Abbildung 73, S. 138). Dieses Design-Pattern ist eine Weiterführung der Designrichtlinie *Informationsexperte*. Dabei wird jede Klasse einer der drei Schichten *Con-*

28 Wird ein Automat per Hardware über logische Gatter und Flip-Flops realisiert, so spart der Mealy-Automat Flip-Flops, da die Zustandsmenge i.d.R. geringer ist. Dies geht jedoch auf Kosten eines etwas aufwendigeren Ausgangsschaltnetzes. Moore- und Mealy-Automaten lassen sich bei gleichem äußeren Verhalten ineinander umformen.

trol, *View* und *Storage* zugeordnet und erhält somit eine Meta-Aufgabe, indem sie entweder eine Steuerungs-, eine (graphische) Darstellungs- oder eine Speicheraufgabe erhält.
Damit wird eine weitere Modularisierung erreicht und das Projekt übertragbarer. So besteht bei der Portierung eines C++-Projekts auf ein anderes Betriebssystem oder auch nur auf eine andere Entwicklungsumgebung das Problem oft darin, dass sich die Bibliotheken, welche die Verwendung der graphischen Oberfläche steuern, unterschiedlich sind. Bei Java besteht das Problem in der Portierung einer Stand-Alone-Anwendung zu einem Applet oder einem Servlet. Durch die konsequente Trennung der Klassen in die 3 Schichten müssen bei einer Portierung nur die Klassen der View-Schicht angepasst werden. Entsprechendes gilt für die Storage-Schicht. Von der Frage der Datenablage in einer Datei oder einer Datenbank ist nur diese Schicht betroffen.
Im Unterricht kann man dieses Thema ansprechen, indem das Klassendiagramm für das Zahlenschloss gesucht wird: Wo soll sich die gefundene und bereits ausformulierte Klasse des Automaten im gesamten Klassendiagramm befinden?
Der erste Schritt zur Trennung der Funktionalität des Programms (Control-Schicht) von der Bedienoberfläche (View-Schicht) wird dabei vollzogen und artikuliert. Dabei sollte auch auf die Vorteile dieses vorerst nur 2-Schichten-Konzepts eingegangen werden: die Portierbarkeit wird verbessert.
Anschließend sollten die Schüler das Design in Programmcode umsetzen (siehe Programm *Zahlenschloss* v1.1, Anhang D und Arbeitsblatt 12, Anhang B.13).
Zur weiteren Vertiefung kann das Projekt *Maier-Suchmaschine* verwendet werden. Eine Maschine zur Suche des regulären Ausdrucks „M[ae][iy]er“[29] soll entwickelt werden. Dadurch bekommen die Schüler einen Zugang zu den recht mächtigen Möglichkeiten regulärer Ausdrücke [El99] und des Weiteren wird das Gelernte vertieft (siehe Programm *maier* v1.1, Anhang D).
Mit der Trennung von View- und Controller-Klassen wird der Weg zum CVS-Pattern (Control, View, Storage) begonnen. Dieser Punkt wird nun durch Hinzufügen von Storage-Klassen vollendet. Die Entscheidung, wo die Daten abgelegt werden (z.B. in einer Datei oder Datenbank) und in welchem Format, wird an diese Klassen delegiert.
Dazu ist wie immer zuerst das Design in Form des Klassendiagramms, nun mit der neuen Storage-Schicht, zu entwerfen und anschließend erst die Programmierung durchzuführen. Umsetzen sollen dies die Schüler mit Hilfe einer persistenten Ablage des Zustands des Zahlenschlosses oder durch Öffnen einer Textdatei für die Maier-Suchmaschine. Je nach vorhandener Zeit kann die Software-Klasse zum Speichern des Zustands und zum Laden des Textes den Schülern ausgeteilt oder mit Hilfe einer verfügbaren Online-Hilfe von ihnen selbst erstellt werden. Da die Zeit aber in der Regel knapp ist, sollte die Programmierung im Fach Computertechnik erfolgen.

Nach Abarbeitung aller vorangegangenen Einheiten ist nun der Weg bereitet, den eigentlich ersten Schritt bei der Problemlösung zu gehen. Die Anwendungsfälle (Use Cases) bieten ein strukturiertes Vorgehen und eine Möglichkeit, im ersten Schritt alle nötigen Informationen zu recherchieren und so den Problemlösungsprozess anzustoßen. Weiterhin dienen sie als Gesprächsgrundlage zwischen dem Softwareexperten und dem Experten des Anwendungsgebiets. Es wird deutlich, dass ein Softwareexperte in der Regel nicht der Experte des Fach-

29 Diese Maschine soll in einem Text zählen, wie oft folgende Formen des Namens Maier vorkommen: Maier, Mayer, Meier, Meyer.

gebiets ist, für die die Software erstellt wird. Deswegen muss intensiv und widerspruchsfrei mit den entsprechenden Anwendungsgebietsexperten kommuniziert werden.
Im Vordergrund steht dabei weniger das Anwendungsfall-Diagramm selbst (siehe Abbildung 74, Anhang A.4.4), als vielmehr die genauere textuelle Beschreibung eines Anwendungsfalls in Form eines Szenarios mit eventuellen Alternativen (siehe Abbildung 75, Anhang A.4.4). Das imaginäre Durchdenken eines Szenarios fördert beim Schüler das detaillierte Vorstellungsvermögen von nacheinander ablaufenden Prozessen.
Umgesetzt wird dies an einem den meisten Schülern bekannten Beispiel, der Registrierkasse. Hier sind die Lernenden sowohl Software- als auch Problembereichsexperten. Trotzdem bringen einige Schüler mehr Wissen mit, da sie z.B. bereits in einem Einkaufszentrum gearbeitet haben. So sind diese die eigentlichen Problembereichsexperten.
Im Vordergrund sollte dabei weniger das Anwendungsfall-Diagramm als vielmehr die genaue Beschreibung der Szenarien mit den typischen Abläufen und den evtl. auftretenden Abweichungen stehen. Wichtig ist dabei ein strukturiertes Vorgehen der Schüler.
Auch dieses Vorgehen wird an einem weiteren Beispiel eingeübt.
Dadurch werden dem Softwarexperten die Anforderungen an das Projekt klarer und der Anwender[30] kann seine Anforderungen genauer artikulieren. Daraus folgt dann das Analyse-Modell (Domain-Model), evtl. unter Zuhilfenahme der Beschreibung einzelner Sequenzen mit dem Sequenzdiagramm. In einem iterativen Prozess (siehe unten) werden die Modelle dann verfeinert.
Mit den Use Cases steht auch ein Hilfsmittel zur Verfügung, um wichtige Programmteile, welche zuerst realisiert werden sollen, zu identifizieren, um so das Gesamtprogramm Stück für Stück zu erhalten. Dies ist auch später für die Schüler von Bedeutung, wenn im Anschluss ein Softwareprojekt in Angriff genommen wird. Mit Hilfe der Use Cases können die Schüler im Projektablauf die Meilensteine, also die Zeitpunkte der Fertigstellung einzelner Softwareteile bzw. Software-Releases, besser planen.

Beim letzten Teil der gesamten Lehrplaneinheit wird der gesamte Entwurfsprozess im Überblick erarbeitet. Dies ist zum einen Teil eine grobe Wiederholung der gesamten Lehrplaneinheit, aber der Ablauf beim Softwareentwurf wird dabei auch strukturiert. Hier sollte mit dem bereits vorhandenen Wissen der Lernenden gearbeitet werden. Es empfiehlt sich dabei, die einzelnen Punkte von den Lernenden auf Kärtchen aufschreiben und dann auf einer Metaplanwand anordnen zu lassen (z.B. siehe Abbildung 76, Anhang A.4.5). Die Betonung auf das iterative Vorgehen und die Unterscheidung der einzelnen Schritte mit ihren Tätigkeiten sollte im Vordergrund stehen.

Anschließend sollten die einzelnen Iterationen auf einer noch höheren Ebene betrachtet werden: Wie stehen die einzelnen Iterationen zueinander, sind in allen Iterationen die einzelnen Schritte gleichwertig?
Der Gang auf diese Metaebene ist zugleich der Sprung zum Unified Process (siehe Abbildung 77, Anhang A.4.5, [Kr99]), der Softwarelebenszyklus wird vorgestellt. Dies ist bereits der erste Schritt zu Projektmanagementmethoden, welche dann in der nächsten Lehrplaneinheit im Rahmen eines Softwareprojekts vertieft werden sollten.

30 Im Unified Process wird der Begriff *Stakeholder* (dt: Interessenvertreter) verwendet. Damit werden alle auch nur im Entferntesten betroffenen Personen angesprochen [Kr99].

In diesem Softwareprojekt können die Lernenden dann die gelernten Verfahren und Prinzipien umsetzen und so den gelernten Stoff vertiefen. Die Problemlösekompetenz wird stark gefördert.

3.7.6 Ausblick Projekt

Anschließend sollte ein Projekt geplant werden, bei dem eine Schülergruppe eine Software erstellt. Dabei kann auf die Methoden und Verfahren des Projektmanagements (siehe LPE 7, Durchführung einer Projektarbeit [It01]) eingegangen werden, damit die Schüler das Projekt erfolgreich und innerhalb der gegebenen Zeit durchführen können. Es sollte dabei darauf geachtet werden, dass die gelernten Verfahren und Methoden richtig eingesetzt werden. Auch in die Bewertung der Projektarbeit sollte dies einfließen.
Das Projekt ist nicht mehr Teil dieser Arbeit, da laut Lehrplan nicht vorgeschrieben ist, dass es sich explizit auf die Lehrplaneinheit 5 *Objektorientierte Analyse und Design* bezieht. Das Projekt kann sich auch auf andere Lehrplaneinheiten beziehen. Damit ist nicht mehr gewährleistet, dass auch in der Kontrollgruppe ein Projekt mit diesen Inhalten durchgeführt wird.

Ein grundlegendes Wissen zum objektorientierten Softwareentwurf wurde vermittelt. Das systematische Herangehen an Probleme wurde eingeübt, was sicherlich auch außerhalb des Softwareentwurfs hilfreich sein wird.

4 Umsetzung und Evaluation des Konzepts

Um das in dieser Arbeit erstellte Konzept zu überprüfen, wurde es in 3 Klassen an 3 verschiedenen Schulen mit insgesamt 63 Schülern umgesetzt. Die Unterrichtsentwürfe der einzelnen Einheiten wurden den teilnehmenden Lehrern rechtzeitig zugeschickt, so dass sie den Unterricht entsprechend gestalten konnten (siehe Kapitel 3.7 und Anhang A bzw. B). Die Entwürfe sind so angelegt, dass noch genügend Freiheit besteht, um z.B. einzelne Punkte in Einzel-, Partner- oder Gruppenarbeit umzusetzen oder Zusätze einzuschieben. Auch die Programmiersprache wurde nicht festgelegt. So wurde in 2 Klassen als Programmiersprache Java in der dritten Klasse C++ eingesetzt.
Als Kontrollgruppe nahmen 3 Schulen mit 4 Klassen und insgesamt 98 Schülern teil[31]. Damit ist ein Vergleich des Unterrichtskonzepts mit anderen Konzepten möglich. In den Kontrollgruppen wurde allerdings nicht ein Konzept, sondern unterschiedliche Konzepte umgesetzt. Diese sind nicht Bestandteil dieser Arbeit, da die Kontrollgruppe nur als Vergleich dienen soll.
Bei allen teilnehmenden Klassen handelt es sich um 12. Klassen des informationstechnischen Gymnasiums in Baden-Württemberg. Alle haben als Voraussetzung den entsprechenden Lehrplan [It01] und damit dieselben Ziele. Auch müssen alle teilnehmenden Schüler nach Ende der 13. Klasse dieselben zentral gestellten Abituraufgaben lösen.

Die Umsetzung und Evaluation erfolgte in einem Feldversuch, d.h. sie erfolgte unter realen Bedingungen. Im Vergleich zu einem Laborversuch sind dabei allerdings die Randbedingungen sehr schlecht zu kontrollieren. Deswegen wurde versucht, diese über zusätzliche Instrumente, wie z.B. einen Fragebogen zum sozialen Umfeld, der Messung der Intelligenz und der Ermittlung der Lernumgebung, zu kontrollieren. Ansonsten bietet der Feldversuch den großen Vorteil, dass die gewonnen Erkenntnisse sich wirklich auf die recht komplexe Realität beziehen.

Für die Untersuchung stellte sich die Frage, ob qualitativ oder quantitativ evaluiert werden sollte.
Eine qualitative Evaluation, wie sie z.B. von Hubwieser, Humbert und Schubert [Sch01] vorgeschlagen wird, bietet sich für explorative Untersuchungen an. Hier wird über Beobachtungen, persönliche Gespräche mit Lehrern und Schülern und über Videos der Unterricht untersucht. Die Ergebnisse bieten meistens einen guten Anstoß für weitere Entwicklungen (explorativ) und Fragen. Unbekannte Phänomene im Untersuchungsgebiet können dadurch gut gefunden werden. Auf der anderen Seite sind die Ergebnisse im strengen, statistischen Sinne wenig repräsentativ, da das Verfahren für eine größere Anzahl von Teilnehmern extrem aufwendig wäre und es dadurch statistisch kaum auswertbar ist.
Bei einer quantitativen Evaluation müssen die Hypothesen bereits vorher feststehen. Feinheiten oder Besonderheiten bei einzelnen Teilnehmern werden nicht berücksichtigt. I.d.R. erfolgt sie über fest vorgegebene Fragebögen, welche von allen Teilnehmern ausgefüllt werden oder über standardisierte Interviews. Sie eignet sich besonders, um vorher postulierte Hypothesen an einer größeren Anzahl von Teilnehmern statistisch abzusichern. Aber ihre Aussagekraft geht meist nicht über die gestellten Fragekomplexe hinaus, d.h. meist ist nur eine Aussage möglich, ob die angenommenen Hypothesen angenommen oder abgelehnt

31 Die Kontrollgruppe bestand aus 4 Klassen, da ein teilnehmender Lehrer 2 Klassen unterrichtete (Schule F).

werden. Allerdings können damit auch Zusammenhänge gefunden werden, sofern die Variablen hierzu bereits vorher bekannt waren und damit in den Entwurf der Fragebögen als Messwerkzeuge eingegangen sind. Der wichtigste Teil der Arbeit bei dieser Art von Evaluation erfolgt vor der eigentlichen Befragung. Die Fragen müssen vorher gut vorbereitet sein. Sollten sich bei der quantitativen Evaluation erst während der Auswertung neue Erkenntnisse zeigen, z.B. durch signifikant große Korrelationen zwischen unterschiedlichen Fragen, welche vorher bei den theoretischen Überlegung nicht vermutet wurden, so bedeutet dies keinen strengen wissenschaftlichen Beweis für diese Erkenntnis. Allerdings kann mit dieser Erkenntnis nun eine neue Hypothese gebildet und dann diese in einer weitergehenden Untersuchung bestätigt oder verworfen werden.

In dieser Arbeit soll ein konkretes Unterrichtskonzept in Bezug zu Bestehenden erforscht und primär das Erreichen der Lernziele untersucht werden. Deshalb und wegen der großen Anzahl von Teilnehmern wird hier die quantitative Evaluation bevorzugt.

Die Hauptthese, dass das hier vorgestellte Konzept in Bezug zu den Lernzielen, wie sie insgesamt im Lehrplan [It01] und in der Lehrzielmatrix (siehe Tabelle 1, S. 26) vorgestellt werden, besser ist als der an den Schulen (Kontrollgruppe) üblicherweise vorhandene Unterricht im Bereich des objektorientierten Softwareentwurfs (Nullhypothese), steht im Vordergrund. Des Weiteren soll untersucht werden, in welchen Bereichen der Lehrzielmatrix die Unterschiede liegen, um ein genaueres Bild des neuen Konzepts zu erhalten. Weiterhin soll der Einfluss des Konzepts auf die Erreichung der Lernziele im Vergleich zu anderen Variablen wie z.B. dem Lehrereinfluss, der Intelligenz oder der Freizeitbeschäftigung mit dem Rechner untersucht werden.
Dazu wurde ein schriftlicher Test als Messwerkzeug entwickelt, welcher die Bereiche der Lehrzielmatrix überprüfen soll (siehe Anhang C.6). Weiterhin wurden der Intelligenzquotient und über einen Fragebogen das soziale Umfeld ermittelt (siehe Anhang C.1).

In der Evaluation der konkreten Umsetzung im Rahmen eines Lehrplans stellen sich dann folgende Fragen:

- Ist das Konzept für die Vermittlung des objektorientierten Softwareentwurfs geeignet bzw. wurden die im Konzept festgelegten Lernziele von den Lernenden erreicht?
- Wo liegen im Vergleich zu einer Kontrollgruppe die Vor- und Nachteile des Konzepts?
- Welche Auswirkung hat das Konzept auf die Erreichung der Lernziele in Bezug auf andere Einflussfaktoren (Lehrereinfluss, Intelligenz, ..)?

Daraus folgen unmittelbar die Hypothesen, welche mit Hilfe der Evaluation geklärt werden sollen:

- Das Konzept ist für die Vermittlung des objektorientierten Softwareentwurfs geeignet.
- Die festgelegten Lernziele (siehe Tabelle 1, S. 26) werden mit Hilfe des Konzepts erreicht.
- Schüler, welche mit Hilfe des Konzepts unterrichtet wurden, sind insgesamt in Bezug auf die Erreichung der Lernziele besser als Schüler, welche nicht mit Hilfe dieses Konzepts unterrichtet wurden.

Schon aus Gründen der höheren Motivation ist bei allem ebenfalls darauf zu achten, dass das Unterrichtskonzept den Lernenden und den Lehrenden Spaß macht. Dies soll ebenfalls untersucht werden, auch wenn sich die höhere Motivation infolge des höheren Spaßes wahrscheinlich in besseren Ergebnissen widerspiegeln wird.

4.1 Durchführung und Messwerkzeuge

Die Durchführung der Evaluation erfolgte im Schuljahr 2002/2003 (siehe Abbildung 22). In der Graphik ist der Ablauf der Unterrichtseinheiten der Umsetzungsgruppe dargestellt. Dies ist für die Kontrollgruppe ohne Relevanz, da dort je nach Lehrer ein anderer Stoffverteilungsplan verwendet wurde. Die einzige Gemeinsamkeit besteht hier zwischen der Umsetzungs- und der Kontrollgruppe im selben, für die Klasse 12 verbindlich vorgeschriebenen Lehrplan.
Da die Stundenplangestaltung an den teilnehmenden Schulen nicht einheitlich war, zog sich diese Phase über das ganze Schuljahr hin. Entscheidende Zeitpunkte waren dabei der Beginn der Lehrplaneinheit und deren Ende, als die entsprechenden Tests und Befragungen erfolgten.

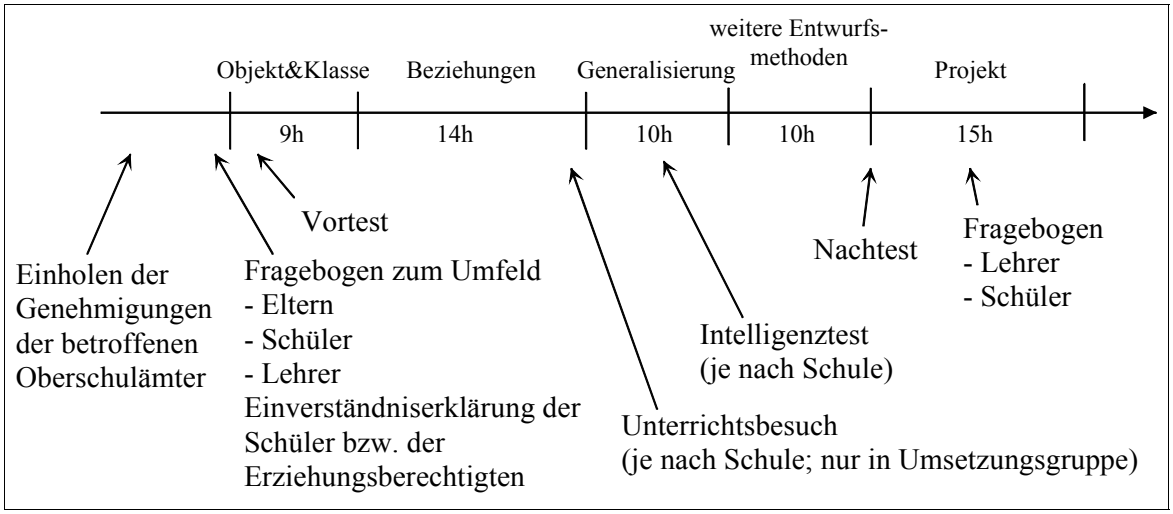


Abbildung 22. Der zeitliche Verlauf der Evaluation im Schuljahr 2002/2003

Durch technische Probleme bedingt konnte der Vortest in der Kontrollgruppe jedoch nicht vor Beginn, sondern erst nach ca. 10 Unterrichtsstunden (je nach Schule) durchgeführt werden. Deswegen sind dessen Ergebnisse nicht sehr aussagekräftig. Es ist jedoch davon auszugehen, dass sich aufgrund ähnlicher Voraussetzungen die Ergebnisse des Vortests der Umsetzungsgruppe an dieser Stelle auch in der Kontrollgruppe wiederfinden.
Der Nachtest erfolgte jedoch in allen Klassen nach Abschluss der Lehrplaneinheit.
Die Genehmigungen der beteiligten Oberschulämter und der Eltern bzw. der Schülerinnen und Schüler für die Evaluation lagen vor (siehe Anhang C.1).

4.1.1 Schüler-, Lehrer- und Elternbefragung

Zur Ermittlung des Schülerumfelds wurden die Schüler und ihre Eltern befragt (siehe Anhang C.1). Dabei wurde das soziale Umfeld über den Beruf der Eltern ermittelt. Der Lernerfolg ist in dieser Lehrplaneinheit unter anderem davon abhängig, ob der Schüler einen Rechner zu Hause nutzen kann und entsprechende Software installiert hat. Auch dies wurde

zusammen mit der Art der Nutzung des heimischen Rechners durch den Schüler in diesem Fragebogen erkundet.

Der von den Lehrern auszufüllende Fragebogen zur Erfassung des Lehrer- und Klassenumfeldes (siehe Anhang C.2) geht auf die fachlichen und pädagogischen Erfahrungen und Kenntnisse des Lehrers und auf die soziokulturellen Rahmenbedingungen der Klasse ein. Neben den Arbeitsjahren als Lehrer im Allgemeinen wurde auch die Tätigkeitsdauer im Bereich der Programmierung und des objektorientierten Softwareentwurfs als Maß für die pädagogische Erfahrung ermittelt. Die fachliche und fachlich-pädagogische Erfahrung zeigt sich an der studierten Fachrichtung und der Fächerkombination des Lehrers. Für schon längere Zeit im Schuldienst stehende Lehrer sind diese Angaben weniger relevant, da dann die persönliche Weiterbildung im Vordergrund steht.
Zusätzlich sollten die Lehrer sich subjektiv in einer Skala von 0 bis 10 in ihrer fachlichen und pädagogischen Kompetenz selbst bewerten.

Für die Rahmenbedingungen der Klasse wurden die Merkmale der Klassengröße und der Anzahl der Unterrichtsstunden im Fach Informationstechnik und Computertechnik erhoben. Der Fragebogen endet mit Fragen zur technischen Ausstattung der Unterrichtsräume sowie zum Verhältnis Rechner zu Schülern und den verwendeten Softwaretools im Unterricht und zur Unterrichtsvorbereitung. Auch diese sind wie im persönlichen Bereich des Schülers wichtige Voraussetzungen für den Lernerfolg.

Am Ende der Lehrplaneinheit wurden die unterrichtenden Lehrer nochmals gebeten, sich selbst subjektiv in ihrer fachlichen und pädagogischen Kompetenz zu bewerten (siehe Anhang C.3 und C.4). Weiterhin wurden weitere subjektive Einschätzungen zum Unterricht abgefragt:

- Wie interessant war der Unterricht?
- Wie war die Motivation der Schüler?
- Wie gut sind die Schüler für das Abitur und ein Studium vorbereitet?

Mit diesen Fragen lässt sich ein grobes Bild der Motivation des Lehrers und seiner Zufriedenheit mit dem Unterricht zeichnen.
Die Lehrer der Umsetzungsgruppe wurden zusätzlich noch zum Konzept befragt, um so die Schwachstellen besser einkreisen zu können.

Auch die Schüler wurden am Ende der Einheit zu ihrer Einstellung in Bezug auf den Softwareentwurf und den Unterricht befragt (siehe Anhang C.5). Hier zeigt sich wie die Schüler den Unterricht subjektiv erlebt haben und wie sie zu einem Beruf in diesem Bereich stehen. Damit die Antworten nicht verfälscht werden, wurden diese Fragen anonym gestellt, so dass nur ein Rückschluss auf die Klasse, nicht aber auf den Schüler, erfolgen kann.

4.1.2 Vor- und Nachtest

Um die Nullhypothese zu überprüfen wurde ein Test entwickelt (siehe Anhang C.6). Als Grundlage diente die Matrix in Tabelle 1 (Seite 26). Laut F. Schott [Sch72] entspricht jedes belegte Feld dieser Matrix einer Aufgabenklasse. Dies würde jedoch im vorliegenden Fall zu 23 Aufgabenklassen führen. Selbst bei einer Aufgabe pro Aufgabenklasse ergeben sich daraus 23 Aufgaben, welches sicherlich zu einer zu starken Belastung der Testpersonen füh-

ren würde. Ein durch die zu lange Prüfungszeit bedingter Leistungsabfall würde das Ergebnis verfälschen. Verstärkt würde dies durch den hohen Zeitbedarf beim Testen der Taxonomiestufen *Synthese* und *Evaluatio*n. Folgende Lösungen sind denkbar:

- Ein Verteilen der Aufgabenklassen auf mehrere Tests, verteilt über einen längeren Zeitraum.
- Nur stichpunktartiges Prüfen einiger Aufgabenklassen, was durch die Abhängigkeiten der Taxonomiestufen erleichtert wird. Die *Anwendung* eines Inhalts setzt auch das *Wissen* und *Verstehen* des Inhalts voraus [Bl56]. Allerdings kann bei einem negativen Ergebnis der Abprüfung der *Anwendung* keine Aussage über die niedrigeren Stufen *Wissen* und *Verstehen* gemacht werden. Sicherlich ist aber bereits in diesem Fall das gesamte Lernziel nicht erreicht.

Aufgrund dieser Überlegungen wird für die Evaluation des in dieser Arbeit vorgestellten Konzepts folgender Weg eingeschlagen:

- Der Vortest soll alle Inhalte, aber nicht in allen Taxonomiestufen abfragen, da davon ausgegangen werden kann, dass selbst die niedrigen Taxonomiestufen den Lernenden im Vortest Schwierigkeiten bereiten. Dadurch wird die Länge des Tests reduziert und die Frustration der Schüler vor Vermittlung der Lehrplaneinheit verringert. Der Eindruck wird abgeschwächt, dass ein zu großer Berg von Inhalten vermittelt wird.
- Im Nachtest wird der Vortest z.T. durch Aufgaben aus den höchsten beiden Taxonomiestufen ergänzt, wobei dadurch weiterhin nicht alle Inhalts-Taxonomiestufen-Kombinationen, wie die Lehrzielmatrix sie vorgibt, verwendet werden. Bei einem Versagen kann dadurch evtl. keine Aussage darüber getroffen werden, welche niedrigere Taxonomiestufe für das Misslingen des Tests verantwortlich ist.

Ausgehend von diesen Überlegungen wurden zu den meisten Zielen Items entworfen (siehe Tabelle 1, S. 26 und Anhang C.6).
Aus technischen Gründen wurde der Vortest für die Kontrollgruppe erst nach ca. 10 Unterrichtsstunden (je nach Schule) vorgelegt, sodass die Aussagekraft für die Kontrollgruppe eingeschränkt ist. Es ist jedoch anzunehmen, dass die Ergebnisse der Umsetzungsgruppe übertragbar sind, da an dieser Stelle keine Unterschiede durch das Konzept auftreten können.
Der Nachtest erfolgte an allen Schulen nach Vermittlung der gesamten Lehrplaneinheit, wobei dieser Zeitpunkt je nach Schule unterschiedlich war, da im Lehrplan die Verteilung der Inhalte auf das Schuljahr dem Lehrer überlassen ist.
Zusätzlich zu den Items wird noch ermittelt, welche subjektive Einstellung der Schüler zum Thema Programmieren besitzt. Da dies sowohl im Vortest als auch im Nachtest erfragt wird, können evtl. auch Änderungen in der Einstellung durch den Unterricht erkannt werden.
Größtenteils sind die Items des Vor- und Nachtests identisch. Für jedes Item ist vermerkt, zu welchem Lernziel es zu rechnen ist. Eine Übersicht des Zusammenhangs zwischen Lernziel und Item liefert Tabelle 8. In dieser Form dauert der Test ca. 90 Minuten.

B	C	E	F	H	I	L	M	O	P	Q	R	U	X	Y
2.1	2.2	3.1	3.2	**5.7**	**5.8**	4.1	4.3	**5.7**	**5.9**	5.4	5.1	5.6	5.3	2.1
2.4	2.3		3.3			4.2	4.4				5.3			2.3
2.5			3.4			4.6	4.5				5.5			3.3
														4.3
30	20	10	30	10	10	30	30	10	10	10	30	10	10	40

Tabelle 8. Der Zusammenhang der Lernziele (Buchstaben, siehe Tabelle 1, S. 26) mit den Items[32] (siehe Anhang C.6)

Jedes Item wird intervallskaliert mit 0-10 Punkten bewertet (0=falsche Lösung, 10=richtige Lösung). Werden innerhalb eines Items mehrere Lernziele abgefragt (z.B. Item 2.1), so wird dieses Item auch mehrfach, jeweils in Bezug auf das entsprechende Lernziel, bewertet. Die Punkte jedes Lernziels werden summiert und gehen so in die Statistik ein. Des Weiteren wird die Gesamtsumme aller Items ermittelt, um so einen ersten Eindruck zu gewinnen. Dieses Vorgehen ist aber problematisch, da dabei implizit davon ausgegangen wird, dass die Lernziele gemäß der maximalen Punktzahl in Tabelle 8 gewichtet sind.
Da aber auch multivariate statistische Verfahren angewandt werden sollen, welche jedes Lernziel als eine Koordinate in einem mehrdimensionalen Raum abbilden, wird das Problem der Gewichtung umgangen. Insofern ist die Ermittlung der Gesamtsumme zum Erkennen erster Ergebnisse nicht so problematisch. Mit den multivariaten Verfahren ist es auch möglich, ein genaueres Bild des Konzepts zu gewinnen. Es ist nicht nur möglich festzustellen, ob das Konzept evtl. besser ist (Nullhypothese), sondern auch, welche Lernziele im Vergleich besonders gut vermittelt wurden.
Die Items, eine mögliche Lösung und die Bewertungsrichtlinien finden sich im Anhang C.6.

4.1.3 Intelligenztest

Um eine Beeinflussung der Messwerte durch die Intelligenz der Schüler zu erfassen, wird von allen Schülern der Intelligenzquotient (IQ) ermittelt.
Der Intelligenzquotient wird als eine über der Population normalverteilte Größe angenommen, wobei der Mittelwert bei $\mu=100$ liegt, mit einer Streuung von $\sigma=15$.
Die Wahrscheinlichkeit, dass eine normalverteilte Größe X in einem Bereich $a < X < b$ liegt, ergibt sich aus der Fläche unter der Dichtefunktion der Normalverteilung $\varphi(x)$. Also

$$P(a<X<b)=\Phi\left(\frac{b-\mu}{\sigma}\right)-\Phi\left(\frac{a-\mu}{\sigma}\right) \qquad (4.1)$$

mit der Verteilungsfunktion

$$\Phi(z)=\int_{-\infty}^{z}\varphi(x)dx \quad , \qquad (4.2)$$

dem Erwartungswert μ und der Streuung σ.
Wählt man für a und b die Werte 90 bzw. 110, so erhält man für die Intelligenzverteilung im Bereich zwischen IQ=90 und IQ=110

$$P(90<X<110)=\Phi\left(\frac{110-100}{15}\right)-\Phi\left(\frac{90-100}{15}\right)=0{,}49\approx 50\% \quad . \qquad (4.3)$$

32 Fett gesetzte Items wurden nur im Nachtest verwendet. Die unterste Zeile gibt die maximale Punktzahl für dieses Lernziel an.

Dies bedeutet, dass 50% der Population einen Intelligenzquotienten im Bereich von IQ=90...110 besitzen und in den Bereichen mit IQ < 90 bzw. IQ > 110 jeweils 25% liegen, da die Normalverteilung in Bezug zum Mittelwert μ symmetrisch ist.

Gemessen wird die Intelligenz mit dem Grundintelligenztest Skala 2 (CFT 20 von R.H. Weiß [We98]). Ausgewählt wurde dieser Test aufgrund seiner Eigenschaft, vor allem die flüssige Intelligenz (General-Fluid-Ability, g_f) zu überprüfen. Darunter versteht man die Fähigkeit, wie gut man sich auf neue Problemstellungen einstellen kann. Dem gegenüber steht die kristallisierte Intelligenz g_c , welche sich vor allem aus Erfahrungen des Individuums speist, oder wie Cattell schreibt [Ca63, S. 268]:

> Die ‚kristallisierte' Intelligenz ist *„die Sammlung gelernter Kenntnisse, die sich ein Mensch angeeignet hat, indem er seine ‚flüssige' Intelligenz beim Lernen in der Schule anwandte. Die ‚kristallisierte' Intelligenz ist gewissermaßen das Endprodukt dessen, was ‚flüssige' Intelligenz und Schulbesuch gemeinsam hervorgebracht haben."*

Im Rahmen des Strukturmodells der Intelligenz nach Jäger [We98, S. 28] kann der Test wie in Tabelle 9 angegeben lokalisiert werden. Hauptsächlich wird dabei die Verarbeitungskapazität bei figuralen Inhalten ermittelt. Aber auch die Bearbeitungsgeschwindigkeit und der Einfallsreichtum werden abgedeckt.

		Inhalte		
		verbal	numerisch	figural
Operationen	Gedächtnis			
	Bearbeitungsgeschwindigkeit			X
	Einfallsreichtum			X
	Verarbeitungskapazität			XX

Tabelle 9. Strukturmodell der Allgemeinen Intelligenz (General Ability) (nach [We98])

Gerade diese Eigenschaft der flüssigen Intelligenz, die Voraussetzung für den Kenntniserwerb zu sein, ist in dieser Untersuchung wichtig. Es ist zu vermuten, dass eine höhere flüssige Intelligenz zu besseren Ergebnissen in den Tests führt, da dann ein größerer Lernerfolg zu erwarten ist.

Die flüssige und die kristallisierte Intelligenz sind jedoch beide altersabhängig (siehe Abbildung 23). Die flüssige Intelligenz g_f nimmt bis zu einem Alter von ca. 20 Jahren zu und fällt dann wieder ab, was bedeutet, dass die Lernfähigkeit mit zunehmenden Alter sinkt.
Die kristallisierte Intelligenz g_c dagegen ist eine streng monoton steigende Größe, wobei sich die Kurve im Alter abflacht. Der Mensch sammelt zunehmend Wissen und Erfahrungen an.

Der Test besitzt 2 Teile, welche unabhängig voneinander die Intelligenz testen, aber auch gemeinsam ausgewertet werden können. Jeder Teil besteht aus 4 Aufgaben mit 46 Items. Die Items sind in Form eines Multiple-Choice-Tests aufgebaut und werden über eine Schablone ausgewertet. Um ein Abschreiben zu verhindern, gibt es 2 Parallelformen, welche sich lediglich in der Reihenfolge der angebotenen Lösungsalternativen unterscheiden. Der Roh-

wert des Tests wird durch Abzählen der richtigen Antworten ermittelt, sodass bei Verwendung beider Tests ein maximaler Rohwert von 92 ermittelt werden kann.
Der IQ ist hier auf ein Alter normiert, d.h. die Definition, dass der mittlere IQ in einer Population $\mu=100$ beträgt, mit einer Streuung von $\sigma=15$, ist altersbezogen. In der praktischen Ausführung werden die erzielten Rohwerte des Tests nur für einen Altersabschnitt auf die Verteilung normiert, sodass für jeden Altersabschnitt extra normiert wird bzw. eine gesonderte Tabelle zur Transformation verwendet wird.

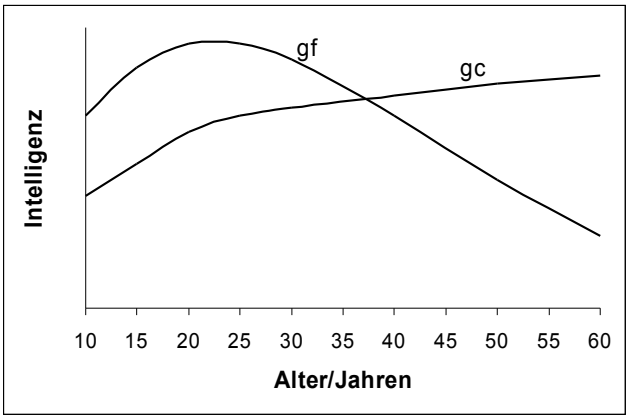


Abbildung 23. Der Verlauf der ‚flüssigen' (gf) und ‚kristallinen' (gc) Intelligenz über das Alter (nach Cattell [Ca68, S. 60])

Diese Tabelle entspricht der Transformationsfunktion, welche die Messungen der Normierungsstichprobe auf die dem Intelligenzquotienten zugrundeliegende Normalverteilung widerspiegelt.
Zur Normierung des Tests wurden für die hier verwendete Altersstufe der 15- bis 19-jährigen N=982 Probanden verwandt. Aus der Auswertung der Normstichproben ergaben sich eine Irrtumswahrscheinlichkeit von 5% für den IQ-Bereich von ±6,6 Punkten. Signifikante Unterschiede zweier Individuen sind erst ab 9 IQ-Punkten ebenfalls bei einer Irrtumswahrscheinlichkeit von 5% festzustellen [We98].

Die Durchführung des gesamten Tests nimmt ca. 60 min in Anspruch.

Da für diese Altersstufe ein breites Altersspektrum von 15 bis 19 Jahren vorliegt und die hier gemessene flüssige Intelligenz altersabhängig ist, wird in Abbildung 24 der Verlauf der Rohwerte über dem Alter für verschiedene Intelligenzen aufgetragen. Die Daten stammen aus [We98], als Alter wird jeweils das mittlere Alter des Altersausschnitts aus den Tabellen gewählt. Es ist zu erkennen, dass bereits ab 15 Jahren die Kurve recht flach verläuft, sodass ein Verfälschen der Messergebisse durch das unterschiedliche Alter der Teilnehmer unerheblich ist. Da alle Teilnehmer aus der 12. Klasse des Gymnasiums stammen, wird das Alter ca. im Bereich 17-19 Jahre streuen. Für die evtl. wenigen Schüler, welche älter als 19 Jahre sind, wird die Tabelle für die 20-29 jährigen verwendet.

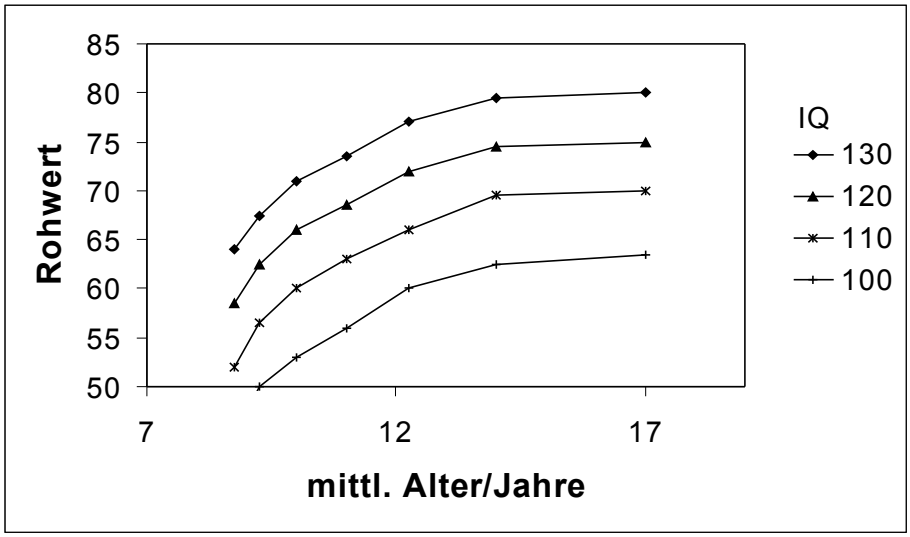


Abbildung 24. Die Rohwerte des Intelligenztests abhängig vom Alter und der Intelligenz. Daten aus [We98]

Abbildung 24 zeigt im Prinzip einen Altersausschnitt aus Abbildung 23, da die Rohwerte ein grobes und unnormiertes Maß für die Intelligenz darstellen. Da in diesem Altersabschnitt die Intelligenz mit steigendem Alter zunimmt, benötigt eine Versuchsperson für denselben IQ mit höherem Alter einen größeren Rohwert.

In diesem Zusammenhang sind die Intelligenz und der Intelligenzquotient (IQ) unterschiedliche Begriffe, da hier der Intelligenzquotient, im Gegensatz zur Intelligenz, auf das Alter normiert ist.

4.1.4 Unterrichtsbesuch

Die Klassen der Umsetzungsgruppe[33] werden während der Durchführungsphase einmal besucht. Diese Besuche sollen einerseits eine Gelegenheit für die Schüler sein, ihre Meinung zu diesem Konzept direkt zu äußern, auf der anderen Seite soll es ein Bild der Umsetzung des Konzepts ergeben.
Da nur eine grobe Unterrichtsplanung an den unterrichtenden Lehrer weitergegeben wird, kann so, exemplarisch an einer Stunde, die Umsetzung im realen Unterricht untersucht werden.

4.1.5 Informatische Vorerfahrungen der Schüler

Um zu ermitteln, wie gut das hier vorgestellte Konzept wirkt, spielen evtl. auch informatische Vorerfahrungen eine Rolle, d.h. welche Fähigkeiten besitzen die Schüler bereits in der Modellierung informatischer Inhalte. Gut geeignet scheint dazu das Modellieren eines einfachen Algorithmus zu sein. Dies soll im Vortest im Rahmen einer kleinen Aufgabe erfolgen. Die Inhalte dazu stammen aus dem Stoff der Klasse 11 [It01][Ai01].

Ein einfacher Algorithmus soll von den Schülern gefunden werden:

33 Es wurden nur zwei Schulen besucht, da die dritte Klasse der Umsetzungsgruppe der Ersteller des Konzepts ist.

In einem Array mit 20 Elementen sind ganze Zahlen (integer) gespeichert. Es soll die Summe über die im Array abgelegten **positiven** Zahlen gebildet werden. Ist also im Array eine negative Zahl (z.B. –5) abgelegt, so soll sie nicht in die Summe mit aufgenommen werden.

Wie die Lösung in Abbildung 25 zeigt, können 4 Bereiche bei der Bewertung der Aufgabe unterschieden werden:

1. Ist der Startwert der Variablen *Summe* auf 0 gesetzt?
2. Ist die Schleife über die Elemente vorhanden?
3. Ist die Auswahl zum Finden der positiven Zahlen korrekt?
4. Ist die Zuweisung der neuen (Teil)summe korrekt?

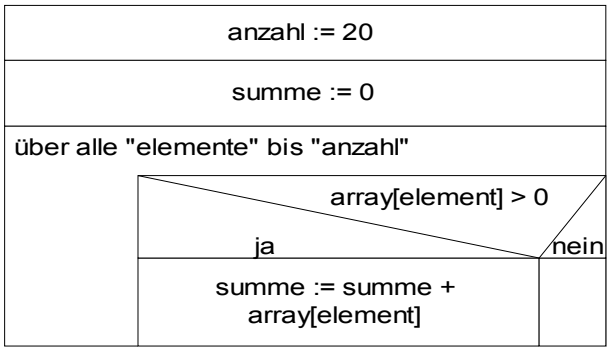


Abbildung 25. Der Lösungsalgorithmus zur ersten Aufgabe des Vortests

Jeder der Bereiche wird mit maximal 2,5 Punkten bewertet, so dass bei dieser Aufgabe maximal 10 Punkte zu erreichen sind.
Ein Vergleich zwischen Umsetzungs- und Kontrollgruppe soll zeigen, dass bei beiden Gruppen dieselbe Vorerfahrung bzgl. der informatischen Modellierung besteht.

4.1.6 Auswertung

Nach Rücklauf der Fragebögen der ca. 170 Schüler mit jeweils ca. 80 Items pro Schüler wurden diese elektronisch erfasst, um so computergestützt die statistischen Auswertungen zu erstellen. Zur Auswertung wurde SPSS in der Version 12.0.1 [spss03] benutzt bzw. zum Teil Excel [ex99] eingesetzt, sofern die Verfahren von SPSS nicht angeboten wurden. Vor allem war dies für die Kontingenztafeln nach Borz [Bo00] und für die verteilungsfreien multivariaten Verfahren nach Zwick [Zw85] nötig (s.u.).

Die Korrektur des Vor- bzw. Nachtests erfolgte in anonymisierter Form. Dazu wurden von einer unabhängigen Person die Lösungsbögen gemischt und mit neutralen Nummern versehen. So war während der Korrektur kein Rückschluss auf einzelne Schüler, Klassen oder die Zugehörigkeit zur Umsetzungs-/Kontrollgruppe möglich. Dadurch konnte die Korrektur in objektiver Weise erfolgen.

Bei der Korrektur der Frage 5.2 (siehe Anhang C.6.1), zeigte sich, dass eine Korrektur nicht möglich war. Die Frage nach der Vorgehensweise bei der Analyse eines Problems wurde von den Schülern zu allgemein verstanden, so dass recht unterschiedliche Antworten, meist außerhalb des objektorientierten Softwareentwurfs genannt wurden. Diese sind jedoch einer

Punktebewertung nicht zugänglich. Folglich wurde diese Frage in der Bewertung ausgelassen.

Leider ist der Anteil der Schülerinnen recht gering. Von allen ca. 170 teilnehmenden Schülern waren insgesamt nur 3 weiblich, so dass hierüber keine Untersuchung erfolgen konnte.

4.2 Schülerumfeld und Rahmenbedingungen

4.2.1 Soziale Herkunft

Um die soziale Herkunft beider Gruppen zu vergleichen, wurde der ausgeübte Beruf des Vaters und der Mutter erhoben. Abbildung 26 zeigt, dass sowohl bei den Berufen des Vaters als auch der Mutter kaum Unterschiede in den beiden Gruppen zu finden sind.

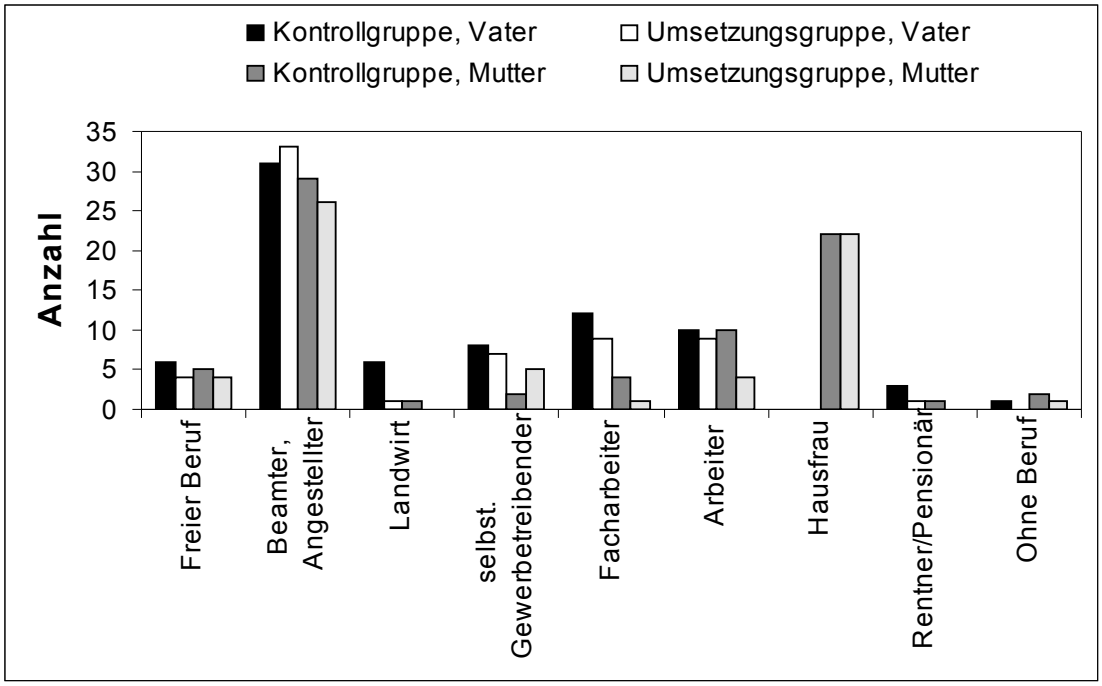


Abbildung 26. Beruf der Eltern

Um das Ergebnis abzusichern, werden die Verteilungen über eine Kontingenztafel verglichen [Bo00,S.158f]. Die ermittelte Kennzahl ist χ^2-verteilt mit einem Freiheitsgrad, abhängig von der Anzahl der Klassifizierungen minus eins. Die Nullhypothese geht bei diesem Verfahren davon aus, dass die beiden Gruppen unterschiedlich sind, so dass bei einer Signifikanz von $\alpha > 5\%$ anzunehmen ist, dass die Gruppen bezüglich der vorgenommenen Klassifizierung gleich verteilt sind.
Die Kennzahl χ^2, die Freiheitsgrade f und die Signifikanz α sind in Tabelle 10 zu finden. Da in der Untersuchung kein Vater Hausmann war, ist bei den Vätern der Freiheitsgrad um eins geringer als bei den Frauen.
Da die Signifikanz α in beiden Fällen recht hoch ist, kann aus dieser Untersuchung geschlossen werden, dass sich die Zusammensetzungen der sozialen Schichten in beiden Gruppen entsprechen. Dies bedeutet, dass der Vergleich zwischen der Umsetzungsgruppe und der Kontrollgruppe unabhängig von der sozialen Schichtung ist.

	χ^2	f	α
Beruf des Vaters	5,43	7	61%
Beruf der Mutter	7,11	8	52%

Tabelle 10. Die Ergebnisse des Vergleichs der Berufe der Eltern über eine Kontingenztafel [Bo00,S.158f]

4.2.2 Häusliche Ausstattung mit Hard- und Software

Ein Vergleich der häuslichen Ausstattung mit Hard- und Software in beiden Gruppen zeigt, dass in der Umsetzungsgruppe im Gegensatz zur Kontrollgruppe zu 100% ein häuslicher Rechner zum Üben vorhanden ist (siehe Abbildung 27). Weiterhin ist in der Umsetzungsgruppe zu 91% eine Programmiersprache wie im Unterricht (Java bzw. C++) vorhanden. Im Gegensatz dazu ist in der Kontrollgruppe bei Vorhandensein eines Rechners nur in 62% der Fälle die Programmiersprache aus dem Unterricht vorhanden. Ein Test über Kontingenztafeln [Bo00,S.158f] ergab für beide Faktoren einen signifikanten Unterschied zwischen den beiden Gruppen. In der Umsetzungsgruppe steht also mehr Teilnehmern ein häuslicher Rechner zur Verfügung und es ist dort auch häufiger eine dem Unterricht entsprechende Programmiersprache installiert.

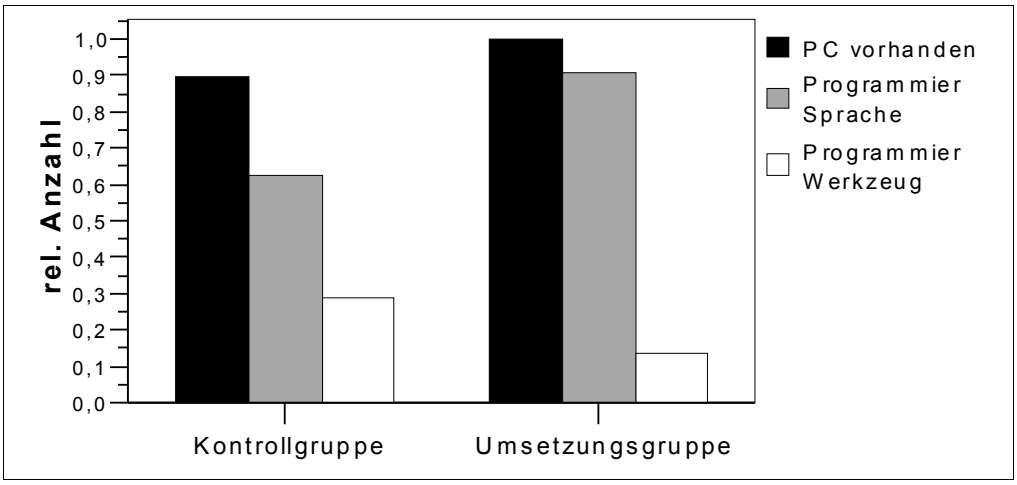


Abbildung 27. Häusliche Ausstattung der Schüler zur Unterrichtsnachbereitung in den beiden Gruppen

Programmierwerkzeuge zum Erstellen von UML-Diagrammen oder Struktogrammen sind in beiden Gruppen nicht sehr häufig vorhanden, allerdings in der Kontrollgruppe häufiger als in der Umsetzungsgruppe. Auch hier ergab ein Test über Kontingenztafeln einen signifikanten Unterschied zwischen den beiden Gruppen.

4.2.3 Beschäftigung in der Freizeit mit dem Rechner

Abbildung 28 zeigt, ob und womit sich die Schüler in der Freizeit mit dem Rechner beschäftigen.

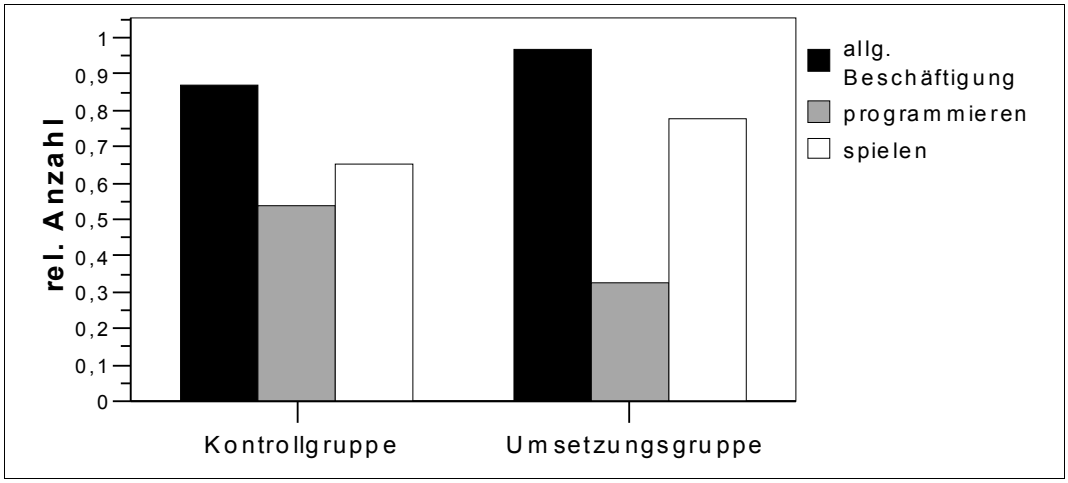


Abbildung 28. Freizeitbeschäftigung mit dem Rechner in beiden Gruppen

Es zeigt sich, dass man sich in der Umsetzungsgruppe signifikant häufiger mit dem Rechner in der Freizeit beschäftigt, wie ein Test über Kontigenztafeln zeigt [Bo00,S.158f].
Allerdings programmieren sowohl in der Kontroll- als auch in der Umsetzungsgruppe recht wenig Schüler in ihrer Freizeit, obwohl eine Programmiersprache zur Verfügung steht. In der Umsetzungsgruppe programmieren z.B. nur 33%, obwohl 91% der Schüler eine Programmiersprache zur Verfügung steht. Ein t-Test und ein Mann-Whitney-Test[34] ergaben jeweils mit einer Restfehlerwahrscheinlichkeit von $\alpha<5\%$, dass signifikant mehr Schüler in der Kontrollgruppe (54%) programmieren. Das Vorhandensein einer Programmiersprache scheint also bei weitem kein Indiz dafür zu sein, dass man sich damit in der Freizeit beschäftigt.
Der Vorteil der Umsetzungsgruppe, dass mehr häusliche Rechner vorhanden und die schulische Programmiersprache installiert ist (s.o.), relativiert sich hier, kehrt sich sogar um, da in der Umsetzungsgruppe weniger programmiert wird als in der Kontrollgruppe.
Gespielt wird in beiden Gruppen ohne signifikanten Unterschied. Im Mittel wird mehr gespielt als programmiert.

4.2.4 Schulische Ausstattung mit Hard- und Software

Tabelle 11 zeigt die Klassengröße, die Anzahl der Rechner und das Verhältnis Schüler pro Rechner. Es zeigt sich, dass zum Teil kleine, auf der anderen Seite aber auch große Klassen, vorhanden sind. Das gleiche gilt auch für die Anzahl der vorhanden Rechner. Interessanter ist das Verhältnis Schüler pro Rechner, welches aufgrund von zeitweisen Klassenteilungen speziell für die Arbeit am Rechner nicht aus den anderen beiden Werten errechnet werden kann. Zum Teil können keine Übungen am Rechner durchgeführt werden, zum Teil hatte jeder Schüler seinen eigenen Rechner.

34 Zum t-Test siehe Kapitel 4.3, zum Mann-Whitney-Test siehe Kapitel 4.3.3.1.

	Klasse	Schüler	Anz. Rechner	Schüler/Rechner
Umsetzungs-gruppe	A	22	12	1,8
	B	29	10	1,5
	C	14	17	1
Kontroll-gruppe	D	29	28	1
	E	30	16	1
	F1	23	0	0
	F2	22	0	0

Tabelle 11. Klassengröße, Anzahl der Rechner und das Verhältnis Schüler pro Rechner

Praktische Arbeit am Rechner ist, um das Programmieren zu erlernen, recht wichtig und es fördert weiterhin stark die Motivation der Schüler. Allerdings ist das Nicht-Vorhanden-Sein eines Rechners nicht so dramatisch, wie dies Tabelle 11 auf den ersten Blick vermuten lässt. Parallel zum Fach Informationstechnik (IT) belegen die Schüler das Fach Computertechnik (CT), in welchem das praktische Programmieren am Rechner geübt werden soll. Dort erfolgte an allen Schulen eine Klassenteilung und der Unterricht findet immer in einem Rechner-Raum statt, so dass jeder Schüler immer an einem Rechner arbeiten kann (genaueres siehe Kapitel 4.2.5). Allerdings erfolgt im Rahmen dieser Untersuchung an zwei Schulen (A, F) dieser Unterricht durch eine andere Lehrkraft, so dass abgestimmter Unterricht erschwert ist.

Als Entwicklungsumgebung wird im Java-Umfeld hauptsächlich Borlands *JBuilder* verwendet. Nur ein Lehrer arbeitet direkt mit der Java-Entwicklungsumgebung von Sun (jdk) unter Zuhilfenahme eines normalen Editors.
Im C++-Umfeld wird Borlands *C++-Builder* benutzt. Allerdings wird nur an einer Schule C++ eingesetzt.
Als weitere Werkzeuge kommen Struktogrammgeneratoren, wie *StrukEd* oder *EasyCode* zum Einsatz. Als Werkzeuge zur Erzeugung der UML-Diagramme kommt *Together* oder *go* zum Einsatz[35].

4.2.5 Anzahl der Unterrichtsstunden

Tabelle 12 zeigt die Anzahl der Stunden, welche für die Lehrplaneinheit *Objektorientierte Analyse und Design* verwendet wurden, zusammen mit dem relativen Anteil der dabei in Klassenteilung bzw. in einem Rechner-Raum abgehaltenen Stunden. Weiterhin zeigt es die Anzahl der Stunden für die Vermittlung der Inhalte zum objektorientierten Softwareentwurf, welche von derselben Lehrkraft zur Unterstützung der Lehrplaneinheit im Fach Computertechnik verwandt wurden, zusammen mit der Gesamtstundenzahl im Fach Informationstechnik (IT) und Computertechnik (CT).

35 JBuilder, C++-Builder, Together[To02]: www.borland.com ; Suns jdk: java.sun.com; StrukEd: www.strukted.de; EasyCode: www.easycode.de; go: Quelle unbekannt.

	Schule	IT Stunden	IT Klassent.	IT Rechner-Stunden	CT Stunden	Stunden
Umsetzungs-gruppe	A	75	0%	100%	0	75
	B	75	50%	50%	25	100
	C	88	0%	100%	22	110
Kontrollgruppe	D	66	0%	33%	44	110
	E	75	25%	25%	30	105
	F1	44	0%	0%	0	44
	F2	44	0%	0%	0	44

Tabelle 12. Die Anzahl der Stunden im Fach IT, der Anteil der Klassenteilung, der Anteil mit vorhandenem Rechner, jeweils in Bezug auf die IT-Stunden, die Anzahl der Stunden im Fach CT und die Gesamtstundenzahl für den objektorientierten Softwareentwurf jeweils beim gleichen Lehrer

Auffällig ist, dass überall mehr unterrichtet wurde als im Lehrplan vorgesehen (45+15 Stunden). Dies zeigt, dass die dort genannten Inhalte in der vorgesehenen Zeit nicht zu bewältigen sind.
In Schule F wurde die geforderte Anzahl von Stunden dagegen unterschritten, was im Vergleich zu den anderen Schulen bedenklich ist.

4.2.6 Fachliche und pädagogische Kompetenz der Lehrer

Die fachliche Kompetenz der Lehrer wurde über die studierte Fachrichtung, der Zeit seit dem Studium und über eine Befragung, woher die Fachkenntnisse stammen, ermittelt. Die pädagogische Kompetenz in der Informatik ergibt sich über die Lehrbefähigung und die Arbeitsjahre als Lehrer.
Ergänzt wird dies durch eine subjektive Selbsteinschätzung für die fachliche und pädagogische Kompetenz.

Tabelle 13 zeigt das Alter der beteiligten Lehrer, die studierte Fachrichtung und die Zeit seit Beendigung des Studiums.
Drei teilnehmende Lehrer haben die Informatik grundständig studiert, zwei in der Kontrollgruppe, einer in der Umsetzungsgruppe.
Im Bereich der Programmierung haben laut Selbstauskunft alle teilnehmenden Lehrer ihre fachlichen Kenntnisse aus der Zeit des Studiums, ergänzt durch Selbststudium oder Lehrerfortbildungen. Einige bringen in diesem Bereich zusätzlich Berufserfahrung aus der Zeit vor der Übernahme in den Schuldienst mit.
Im Bereich des objektorientierten Softwareentwurfs erwarb die Kontrollgruppe ihre Kenntnisse durch Studium oder Berufserfahrung, in der Umsetzungsgruppe überwiegt das Selbststudium.
Das Alter der Lehrer ist mit jeweils einer Ausnahme pro Gruppe nahezu identisch.

	Schule	Alter	Studium	Studium Ende	Prog.	Obj. Softwareentw.
Umsetzungsgruppe	A	30..40	El	12	S,E	E
	B	50..60	Ph	28	S,E	E,L
	C	30..40	Inf	6	S,B,E	B,E
Kontrollgruppe	D	40..50	MB	14	S,B,E	B,E
	E	30..40	WInf	9	S,L	B,E
	F	30..40	Inf	5	S,B	S,E

Tabelle 13. Die fachlichen Voraussetzungen der Lehrer an den Schulen[36]

Einen Blick auf die pädagogischen Voraussetzungen der Lehrer zeigt Tabelle 14. In der Umsetzungsgruppe sind die Lehrer mit einer längeren Dienstzeit zu finden. Dies zeigt sich auch in der Dauer ihrer pädagogischen Arbeit im Bereich der Programmierung. Im Bereich des objektorientierten Softwareentwurfs sind jedoch mit einer Ausnahme die Dauer der pädagogischen Erfahrungen bei allen gering. In der Fächerkombination der Lehrbefähigung der Lehrer zeigt sich wieder die Heterogenität der Fächer, wie dies bereits beim Studium festzustellen war.

	Schule	Dienstzeit	Fach1	Fach2	Prog.	Obj. Softwareentw.
Umsetzungsgruppe	A	8	Nt	Et	5	5
	B	24	M	Ph	20	1
	C	3	Inf	Nt	10 (an Uni)	0
Kontrollgruppe	D	9	Inf	Ft	7	2
	E	2	Dv	M	1	0
	F	1	Inf	M	1	0

Tabelle 14. Die pädagogischen Voraussetzungen der Lehrer an den Schulen[37]

Nur ein Lehrer in der Umsetzungsgruppe bringt in seiner Fächerkombination das Fach Informatik mit. Die beiden anderen bringen afine Fächer wie die Mathematik oder die Nachrichtentechnik mit. Im Gegensatz hierzu sind alle Lehrer in der Kontrollgruppe in ihrem Fach beschäftigt.

Alle beteiligten Lehrer wurden vor Vermittlung der Einheit gebeten, eine Einschätzung ihrer fachlichen und pädagogischen Kompetenz abzugeben. Dazu konnten Sie in einer Skala von 0 (keine Kenntnisse/Erfahrungen) bis 10 (sehr gute Kenntnisse/Erfahrungen) in den Bereichen Programmierung und objektorientiertem Softwareentwurf jeweils pädagogisch und fachlich ihre subjektive Selbsteinschätzung abgeben.

36 Studium: El – Elektrotechnik, Inf – Informatik, MB – Maschinenbau, Ph – Physik, WInf – Wirtschaftsinformatik; Studium Ende: Zeit in Jahren seit Abschluss des Studiums; Prog. und Obj. Softwareentw.: B – Berufserfahrung, E – Selbststudium, L – Lehrerfortbildung, S – Studium.

37 Fach: Dv – Datenverarbeitung, Et – Energietechnik, Ft – Fertigungstechnik, Inf – Informatik, M – Mathematik, Nt – Nachrichtentechnik, Ph – Physik.

Diese sind nur als subjektiv anzusehen und aufgrund der geringen Stichprobe nicht repräsentativ.

		Umsetzungsgruppe		Kontrollgruppe	
		Mittelwert	Einzelwerte	Mittelwert	Einzelwerte
fachlich	Programmieren	8,2	6; 9; 9,5	6,7	8; 3; 9
	Obj. Soft-wareent.	6,3	9; 5; 5	4,3	5; 0; 8
päd.	Programmieren	7,7	6; 9; 8	6,7	8; 5; 7
	Obj. Soft-wareent.	5,7	8; 4; 5	4,7	8; 0; 6

Tabelle 15. Die subjektive Selbsteinschätzung der Lehrer vor Vermittlung der Unterrichtseinheit

4.3 Ergebnisse

Zum Vergleich der Umsetzungs- mit der Kontrollgruppe wird in dieser Arbeit häufig der t-Test eingesetzt. Bei einem t-Test können 2 Gruppen miteinander verglichen werden [Bo93, S. 132ff]. Es wird dabei die Nullhypothese $H_0: \mu_1 - \mu_2 = 0$ gegen die Alternativhypothese $H_1: \mu_1 - \mu_2 \neq 0$ gesetzt. Wobei μ_1 und μ_2 die tatsächlichen Mittelwerte in den beiden Populationen darstellen.

Da jedoch nur eine Stichprobe aus den beiden Grundgesamtheiten der Populationen gezogen werden kann, streut der gemessene Mittelwert um den tatsächlichen Mittelwert. Allerdings kann durch Schätzung des Standardfehlers, ausgehend von den Messwerten, die Verteilung der Differenz der tatsächlichen Mittelwerte geschätzt werden.

Es kann jedoch vorkommen, dass die tatsächlichen Mittelwerte falsch geschätzt werden, also die Nullhypothese H_0 abgelehnt wird, obwohl sie korrekt ist. Dies ist der α-Fehler (Fehler 1. Art).

Sofern nicht anders erwähnt, wird in der gesamten Arbeit davon ausgegangen, dass ein α-Fehler $\alpha < 5\%$ (Signifikanzniveau) so gering ist, dass das Ergebnis signifikant ist und die Nullhypothese sicher abgelehnt werden kann, d.h. es bleibt eine Wahrscheinlichkeit kleiner 5%, dass die Nullhypothese trotzdem gilt.

Aus dem oben genannten lässt sich eine Messgröße *t* ermitteln. Daraus kann dann mit Hilfe der Studentschen t-Verteilung der α-Fehler ermittelt werden. Wegen des zentralen Grenzwertsatzes nähert sich die Studentsche t-Verteilung bei größeren Stichproben (n > 50) der Normalverteilung an.

Allerdings müssen die Voraussetzungen für den t-Test erfüllt sein [Bo93, S.133]:

1. Für kleine Stichprobenumfänge müssen die Messwerte normalverteilt sein.
2. Wenn die Varianzen in den beiden Gruppen unterschiedlich sind, muss der Freiheitsgrad der t-Verteilung korrigiert werden.

4.3.1 Intelligenztest

Ein Vergleich des Intelligenzquotienten zwischen den beiden Gruppen zeigt keinen signifikanten Unterschied. Vergleicht man jedoch die einzelnen Klassen (siehe Tabelle 16), so sticht die Schule D mit einem mittleren IQ von 131 aus den anderen hervor. Bemerkenswert

ist weiterhin, dass in dieser Klasse der niedrigste gemessene Wert bereits sehr nahe zum Mittelwert der anderen Klassen ist. Auch der höchste gemessene Wert liegt in dieser Klasse. Für die restlichen Schulen liegen die Mittelwerte im Bereich von 119-122. Der Mittelwert über alle Schüler beträgt IQ=122.

	Schule	N	Minimum	Maximum	Mittelwert	Standard-abweichung
Umsetzungsgruppe	A	21	104	132	121,3	8,7
	B	27	82	146	119,0	14,6
	C	12	102	146	121,7	13,5
Kontrollgruppe	D	23	118	149	131,3	9,3
	E	24	96	138	121,5	10,8
	F1	22	92	146	121,1	13,7
	F2	21	99	134	121,5	8,2

Tabelle 16. Vergleich der IQ-Werte in den einzelnen Klassen

4.3.2 Ermittlung des Lernerfolgs

Um den Lernerfolg messen zu können, mussten alle Schüler einen Vor- und Nachtest ablegen. Aus den Unterschieden der beiden Tests kann dadurch auf den Lernerfolg geschlossen werden. Zur statistischen Auswertung eignet sich ein t-Test mit gepaarten Messwerten [Bo93], welcher als Messwert die Differenz zwischen den beiden Tests verwendet. Dazu werden für jeden Schüler die Ergebnisse innerhalb der einzelnen Lernziele[38] summiert und die Differenz aus dem Ergebnis des Nachtests mit dem Vortest als Messgröße verwendet. Zusätzlich wird dies auch für die Differenz der Gesamtergebnisse der beiden Tests durchgeführt.

In Abbildung 29 ist für die beiden Gruppen für jedes Lernziel die mittlere Differenz zwischen Nach- und Vortest angegeben. In Tabelle 17 ist das Ergebnis des Gesamttests zu finden.

Ein t-Test mit gepaarten Messwerten ergibt für beide Gruppen und für alle Lernziele, mit einer Ausnahme, eine signifikante Verbesserung der Ergebnisse im Nachtest in Bezug auf den Vortest. Nur in der Kontrollgruppe ergibt sich für das Lernziel Q keine Verbesserung, sowohl im Vor- als auch im Nachtest wurde diese Frage nicht richtig beantwortet. Dieses Lernziel bezieht sich auf die Frage 5.4, bei welcher Designrichtlinien abgefragt wurden.

In beiden Gruppen wurde also, mit der Ausnahme des Lernziels Q (Klassenentwurf Wissen, siehe Tabelle 1, S. 26) in der Kontrollgruppe, über alle Lernziele ein Lernerfolg erreicht.

Ein t-Test zum Vergleich der Verbesserung der Ergebnisse zwischen den beiden Gruppen ergibt bis auf das Lernziel X (UML Wissen; siehe Tabelle 1 auf S. 26) eine jeweils signifikant stärkere Verbesserung in der Umsetzungsgruppe als in der Kontrollgruppe.

38 Lernziele siehe Tabelle 1 auf S. 26.

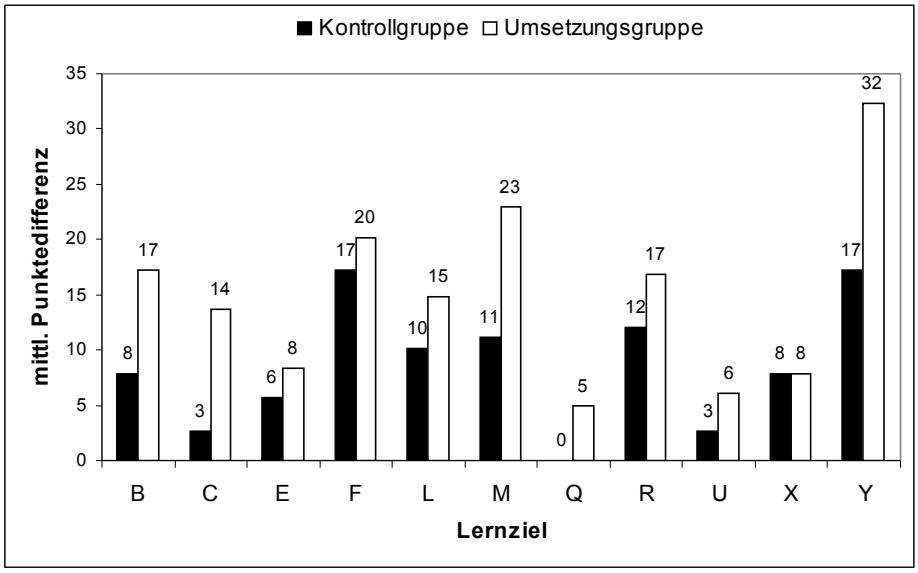


Abbildung 29. Mittlere Abweichung der Ergebnisse für die einzelnen Lernziele in den beiden Gruppen zwischen Nach- und Vortest

Dies bedeutet, dass in der Umsetzungsgruppe die Ergebnisse stärker verbessert wurden als in der Kontrollgruppe, oder anders ausgedrückt, der Lernerfolg in der Umsetzungsgruppe ist signifikant größer als in der Kontrollgruppe.

	N	mittl. Differenz Nachtest - Vortest	Std. Abweichung
Kontrollgruppe	92	113,9	36,2
Umsetzungsgruppe	48	193,2	45,0

Tabelle 17. Mittlere Abweichung der Ergebnisse in den beiden Gruppen zwischen Nach- und Vortest

4.3.3 Vergleich des Nachtests in den beiden Gruppen

4.3.3.1 Univariate Analyse

Für einen einfachen Vergleich zwischen Umsetzungs- und Kontrollgruppe werden die Punkte der einzelnen Testitems des Nachtests addiert (siehe auch Kapitel 4.1.2).
Ein Vergleich der Mittelwerte zeigt, dass die Umsetzungsgruppe besser ist (siehe Tabelle 18). Weiterhin sind auch die Extremwerte sowohl im unteren als auch im oberen Wert in der Umsetzungsgruppe besser.
Etwas genauer zeigt dies Abbildung 30, dort sind in einem Boxplot die beiden Gruppen gegenübergestellt. Neben dem Mittelwert ist auch der Bereich, in dem sich 50% der Teilnehmer befinden, gekennzeichnet (25% bzw. 75% Percentile). Des Weiteren sind dort auch das Minimum und das Maximum der Ergebnisse gekennzeichnet. Überall liegen die Ergebnisse der Umsetzungsgruppe deutlich höher als in der Kontrollgruppe.

	N	Minimum	Maximum	Mittel	Std. Abweichung
Kontrollgruppe	98	51,3	248,5	188,798	36,4530
Umsetzungsgruppe	63	96,5	279,1	217,903	43,12

Tabelle 18. Die Summe der Punkte im Nachtest im Vergleich der beiden Gruppen

Interessanterweise liegt der maximale Wert in der Kontrollgruppe (248,5) noch knapp unter der 75% Percentile (251,5) der Umsetzungsgruppe, d.h. 25% der Teilnehmer aus der Umsetzungsgruppe erreichten ein besseres Ergebnis als der Beste der Kontrollgruppe.

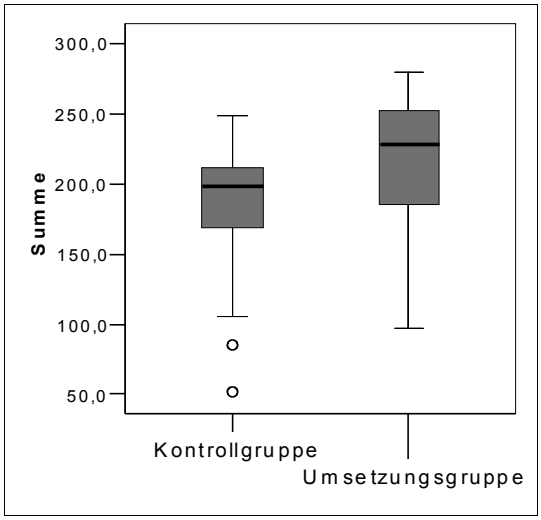


Abbildung 30. Boxplot des Ergebnisses des Nachtests. Die beiden Ausreißer liegen mehr als das 1,5-fache unter der 25%-Percentile in Bezug zur Weite der mittleren 50%

Allerdings überschneiden sich die Ergebnisse beider Gruppen, sodass die Ergebnisse dieser beschreibenden Statistik nicht sehr aussagekräftig sind.
Deswegen wird mit einem t-Test das Gesamtergebnis des Tests verglichen (siehe Tabelle 19). Der Freiheitsgrad wird dabei korrigiert, da die Varianzen in den beiden Gruppen unterschiedlich sind (siehe Kapitel 4.3).

t	df	Sig. (2-seitig)	Mittlere Differenz	Std. Fehler Differenz	95% Konfidenz-Intervall der Differenz	
4,44	116,4	0,000	29,1	6,56	16,1	42,1

Tabelle 19. Ergebnis des t-Tests zum Vergleich der Mittelwerte der Gesamtpunktzahl in den beiden Gruppen

Mit deutlicher Signifikanz (Sig. $\alpha = 0{,}000$) ist ein Unterschied zwischen den beiden Gruppen erkenntlich. Mit 95% Wahrscheinlichkeit ist die Umsetzungsgruppe ca. 16 bis 42 Punkte besser als die Kontrollgruppe, im Mittel sind die Teilnehmer der Umsetzungsgruppe 29 Punkte besser. Auch unter der weniger strengen Annahme, dass die Varianzen in beiden Gruppen identisch sind, ergibt sich nahezu dasselbe Bild.

Für die Durchführung des t-Tests sollte die Gesamtsumme normalverteilt sein (siehe Kapitel 4.3). Dies wird mit dem Kolmogorov-Smirnov- und Shapiro-Wilk-Test untersucht [spss03] (siehe Tabelle 20), welche als Nullhypothese die Normalverteilung annehmen.
Beide Tests ergeben das gleiche Bild: Die Hypothese, dass die Gesamtsumme normalverteilt ist, wird verworfen (Sig. < 5%), so dass die Normalverteilungs-Voraussetzung verletzt ist.

	Kolmogorov-Smirnov			Shapiro-Wilk		
	Statistik	df	Sig.	Statistik	df	Sig.
Kontrollgruppe	0,141	98	0,000	0,929	98	0,000
Umsetzungsgruppe	0,117	63	0,031	0,944	63	0,006

Tabelle 20. Der Test auf Normalverteilung für die Gesamtsumme des Nachtests. Der Kolmogorov-Smirnov-Test wurde mit Hilfe der Lillefors Signifikanz korrigiert [spss03].

Laut Borz [Bo93, S.133] gilt für den t-Test, dass „*der Test eher zugunsten der H1 [entscheidet], wenn die Varianz in der kleineren Stichprobe größer ist als die Varianz in der größeren Stichprobe*“. Dies ist hier der Fall, so dass die Gefahr besteht, dass die Alternativhypothese H1 (hier: die Gruppen sind unterschiedlich) angenommen wird, obwohl sie falsch ist.

Um das Ergebnis, dass die beiden Gruppen unterschiedlich sind, abzusichern, wird deswegen auch das verteilungsfreie Verfahren nach Mann-Whitney durchgeführt [Bo00, S. 200ff]. Dabei werden die Messwerte aller Teilnehmer in eine Rangfolge gebracht, wobei der kleinste Wert den 1. Rang erhält. Im weiteren wird nur noch mit diesem Rang gearbeitet, so dass es unwichtig ist, ob die Messwerte normalverteilt sind.
Anschließend wird für jeden Teilnehmer ausgezählt, wie viele Teilnehmer der anderen Gruppe einen schlechteren Rang haben. Dieser Wert wird jeweils für beide Gruppen getrennt über alle Teilnehmer summiert und ergibt die Werte U bzw. U'. Der kleinere der beiden Werte wird nun als Prüfgröße U gewählt.
Mit Hilfe der Permutationsrechnung lässt sich nun eine Wahrscheinlichkeit angeben, wie viele Kombinationen es gibt, die diesen U-Wert unterschreiten. Daraus lässt sich dann der α-Fehler ermitteln.

Für größere Stichproben ist U approximativ normalverteilt mit dem Erwartungswert

$$E(U)=\frac{N_1 \cdot N_2}{2} \tag{4.4}$$

mit N_1 bzw. N_2 gleich der Anzahl der Teilnehmer in den beiden Gruppen, und der Streuung

$$\sigma_U=\sqrt{\frac{N_1 \cdot N_2 \cdot (N_1+N_2+1)}{12}}\ , \tag{4.5}$$

so dass die Permutationsrechnung entfallen kann [Bo00, S203].

Tabelle 21 zeigt das Ergebnis nach Mann-Whitney. Auch dieses zeigt mit sehr hoher Signifikanz (Asymp. Sig.$\alpha = 0{,}000$; Approximation der Normalverteilung), dass die Gruppen unterschiedlich sind.

Mann-Whitney U	1779,000
Normalverteilung Z	-4,531
Asymp. Sig. (2-seitig)	0,000

Tabelle 21. Das Ergebnis des Vergleichs der beiden Gruppen nach Mann-Whitney

Dass auch hier die Umsetzungsgruppe die besseren Ergebnisse liefert, zeigt Tabelle 22. Dort ist der mittlere Rang in der Umsetzungsgruppe größer als in der Kontrollgruppe, d.h. die Werte besser, da bei diesem Verfahren ein geringerer Wert (hier die Gesamtpunktzahl) einen kleineren Rang bedeutet.

	N	Mittlerer Rang	Rangsumme
Kontrollgruppe	98	67,65	6630,00
Umsetzungsgruppe	63	101,76	6411,00
Total	161		

Tabelle 22. Die mittleren Ränge in den beiden Gruppen

4.3.3.2 Multivariate Analyse

Ein Nachteil des oben stehenden Verfahrens besteht darin, dass die Ergebnisse der einzelnen Items einfach addiert werden und so das Gesamtergebnis eines Teilnehmers ermittelt wird. Wie bereits oben erwähnt erfolgt dabei implizit eine Gewichtung der Lernziele. Um dieses Problem zu umgehen, wird auch multivariat getestet. Dabei betrachtet man jeden Messwert (hier: jedes Lernziel) als eine eigene Dimension und erhält so eine n-dimensionale statistische Auswertung. Als Faktor verwendet man hier nur die Mitgliedschaft in einer der beiden Gruppen (Umsetzungsgruppe, Kontrollgruppe), so dass eine einfaktorielle multivariate Varianzanalyse verwendet werden kann.

Da mehrdimensional gearbeitet wird, erfolgt die Berechnung über Matrizen. Dabei spielt einerseits die Hypothesenmatrix H, andererseits die Fehlermatrix E eine wichtige Rolle [Bo93].
Bei der Erstellung der zur Diagonalen symetrischen Hypothesenmatrix

$$H=\begin{pmatrix} h_{11} & \dots & \dots & h_{1k} \\ \dots & h_{ii} & \dots & \dots \\ \dots & h_{ii'} & \dots & \dots \\ h_{k1} & \dots & \dots & h_{kk} \end{pmatrix} \tag{4.6}$$

mit k als der Anzahl der Dimensionen, hier also der Anzahl der Lernziele, wird davon ausgegangen, dass innerhalb einer Gruppe (hier z.B. innerhalb der Kontrollgruppe) keine Abweichungen aufgetreten sind, sondern nur durch das unterschiedliche Treatment zwischen den Gruppen. Es wird also angenommen, dass die Unterschiede nur durch die hier unterschiedliche Beschulung entstanden sind.

Ausgehend von dieser Überlegung werden die Diagonalelemente der Matrix nach

$$h_{ii}=n\sum_j(\overline{A_{ij}}-\overline{G_i})^2 \qquad (4.7)$$

berechnet. Dabei ist $\overline{A_{ij}}$ der Mittelwert des i.Lernziels (z.B. Lernziel A) und des j.Treatments (z.B. der Kontrollgruppe), $\overline{G_i}$ der Mittelwert des i.Lernziels über alle Treatments und *n* die Anzahl aller Messwerte des i.Lernziels. h_{ii} ist also ein Maß für die quadrierte Abweichung des Mittelwerts innerhalb eines Treatments vom Mittelwert im Ganzen. Gleichung (4.7) läßt sich umrechnen zu [Bo93]

$$h_{ii}=\sum_j\left(\frac{A_{ij}^2}{n_j}\right)-\frac{G_i^2}{N} \quad , \qquad (4.8)$$

mit A_{ij} der Summe der Messwerte des i.Lernziels und des j.Treatments, n_j als der Anzahl aller Messwerte im j.Treatment, G_i als der Summe der Messwerte des i.Lernziels und *N* als der Anzahl aller Messwerte, also

$$N=\sum_j n_j \quad . \qquad (4.9)$$

Für die restlichen Elemente der Matrix außerhalb der Diagonalen wird das Quadrat durch das Produkt der beiden Summen der Indizes i und i' ersetzt, also

$$h_{ii'}=\sum_j\left(\frac{A_{ij}A_{i'j}}{n_j}\right)-\frac{G_iG_{i'}}{N} \quad . \qquad (4.10)$$

Dies ähnelt dem Vorgehen bei der Entwicklung der Kovarianz ausgehend von der Varianz. So ist h_{ii} mit der Varianz und $h_{ii'}$ ($i\neq i'$) mit der Kovarianz verwandt.

Um die Fehlermatrix E zu bilden, wird davon ausgegangen, dass alle Abweichungen nur innerhalb einer Gruppe aufgetreten sind. Darin sind nun alle Störungen zu finden, welche nicht berücksichtigt wurden, wie z.B. der Einfluss des Wetters auf die Schülerleistungen. Die Elemente der quadratischen Fehlermatrix

$$E=\begin{pmatrix} e_{11} & \dots & \dots & e_{1k} \\ \dots & e_{ii} & \dots & \dots \\ \dots & e_{ii'} & \dots & \dots \\ e_{k1} & \dots & \dots & e_{kk} \end{pmatrix} \qquad (4.11)$$

berechnen sich nun nach

$$e_{ii}=\sum_j\sum_m(x_{mij}-\overline{A_{ij}})^2 \quad , \qquad (4.12)$$

mit x_{mij} dem m.Messwert des i.Lernziels und des j.Treatments und A_{ij} als dem Mittelwert der Messwerte des i.Lernziels, des j.Treatments. e_{ii} ist also ein Maß für die quadrierte Abweichung der einzelnen Messwerte vom Mittelwert innerhalb des Treatments. Gleichung (4.12) lässt sich wieder umrechnen [Bo93] zu

$$e_{ii}=\sum_j\sum_m x_{ijm}^2-\sum_j\left(\frac{A_{ij}^2}{n_j}\right) \quad . \qquad (4.13)$$

Ähnlich wie oben werden für die Elemente außerhalb der Diagonalen die Quadrate durch Produkte ersetzt, so dass sich

$$e_{ii'}=\sum_j \sum_m x_{ijm} x_{i'jm} - \sum_j \left(\frac{A_{ij} A_{i'j}}{n_j}\right) \tag{4.14}$$

ergibt.

Addiert man die Hypothesenmatrix *H* und die Fehlermatrix *E*, so ergibt sich eine Matrix, welche die gesamte Varianz repräsentiert [Bo93, S. 548 ff]:

$$D_{tot}=H+E \tag{4.15}$$

Aus der Hypothesenmatrix *H* und der Fehlermatrix *E* kann nun auf unterschiedliche Weise eine neue Matrix gebildet werden, welche das Verhältnis zwischen dem Treatment-Anteil im Bezug zum Fehler-Anteil darstellt. Die Eigenwerte bzw. deren Summen (Spur) oder Produkte sind dann approximativ F-verteilt, so dass damit ein Messwert auf statistische Signifikanz untersucht werden kann.
Eine Zusammenfassung der Verfahren zeigt Tabelle 23.

Verfahren	**Matrix**	**Wert**
Pillai-Spur	$H(H+E)^{-1}$	$\sum e_i$
Wilks-Lamda	$E(H+E)^{-1}$	$\prod e_i$
Hotteling-Spur	HE^{-1}	$\sum e_i$
größte charakteristische Wurzel nach Roy	HE^{-1}	$max\, e_i$

Tabelle 23. Die multivariaten Verfahren mit der verwendeten Matrix und der Methode zur Ermittlung der Messgröße [Bo93, S. 548 ff]. e_i ist jeweils ein Eigenwert der angegebenen Matrix.

Tabelle 24 zeigt das Ergebnis des Vergleichs der beiden Gruppen mit den multivariaten Methoden. Der Signifikanzwert von 0,000 zeigt deutlich die Unterschiedlichkeit der beiden Gruppen auch unter multivariater Ermittlung. Allerdings ergibt ein Blick auf den Levene-Test auf Gleichheit der Fehlervarianzen, dass bei den meisten Lernzielen unterschiedliche Varianzen in den beiden Gruppen vorhanden sind. Die Gleichheit der Fehlervarianzen in den Gruppen ist jedoch eine wichtige Voraussetzung für diesen Test [spss03].
Deswegen werden die beiden Gruppen auch mit einem verteilungsfreien multivariaten Verfahren verglichen. Verwendet wird der Rechenweg von R. Zwick [Zw85], welcher wiederum auf den Ansätzen von Puri und Sen [Pu71] basiert.
Dazu werden die Messwerte – also die Ergebnisse aufgegliedert nach Lernzielen – im ersten Schritt in Rangwerte umgerechnet, die Messwerte also ihrer Größe nach sortiert. Diese Rangwerte werden dann als Ausgangswerte für die Ermittlung der Kennziffern nach Pillai-Spur *PS* verwendet. Allerdings wird diese Kennziffer laut R. Zwick dann nicht nach der F-Statistik, sondern nach der χ^2-Statistik ausgewertet.

	Wert	F	Hypothese df	Fehler df	Signifikanz
Pillai-Spur	1,638	44,067	30,000	292,000	0,000
Wilks-Lambda	0,01	123,26	30,000	290,000	0,000
Hotelling-Spur	66,417	318,801	30,000	288,000	0,000
größte charakteristische Wurzel nach Roy	64,531	628,11	15,000	146,000	0,000

Tabelle 24. Ergebnis des multivariaten Tests mit dem Faktor der Zugehörigkeit zur Umsetzungs-/Kontrollgruppe

Der Wert errechnet sich nach

$$\chi^2=(N-1)\cdot PS \quad \text{(nach [Zw85])}, \tag{4.16}$$

wobei N hier die Anzahl der Versuchsteilnehmer in beiden Gruppen darstellt. Als Freiheitsgrad *df* wird die Dimension des n-dimensionalen Raumes, also die Anzahl der abhängigen Variablen, hier der Lernziele, verwendet.
Die Rangermittlung und die Berechnung der Kennziffer von Pillai-Spur *PS* erfolgt mit Hilfe von SPSS [spss03]. Dabei ergibt sich für Pillai-Spur *PS = 1,517* bei *N=161* Versuchsteilnehmern. Es berechnet sich nun

$$\chi^2=(161-1)\cdot 1{,}517=242{,}72 \quad . \tag{4.17}$$

Bei *df=15* Freiheitsgraden (15 Lernziele) ergibt dies eine Signifikanz s von

$$s=1-p=1-\int_0^{242{,}72} f_{15}(\chi^2)\,d\chi^2=0{,}000 \tag{4.18}$$

(p berechnet mit SPSS [spss03][39]), mit $f_{15}(\chi^2)$ der χ^2-Verteilung mit 15 Freiheitsgraden.

Dies bedeutet, dass das festgestellte bessere Abschneiden der Umsetzungsgruppe auch multivariat, d.h. ohne die Berücksichtigung einer Gewichtung der Lernziele sowohl unter der Annahme der Normalverteilung also auch ohne diese Annahme bestätigt wird.

4.3.4 Analyse der Gruppenunterschiede

Es soll nun untersucht werden, welche Lernziele für das oben festgestellte bessere Abschneiden der Umsetzungsgruppe verantwortlich sind. Dazu werden mit Hilfe eines t-Tests die Ergebnisse in den einzelnen Lernzielen verglichen. In Tabelle 25 sind die Ergebnisse dargestellt. Die eingetragenen Werte geben an, inwieweit die Umsetzungsgruppe besser als die Kontrollgruppe ist (mittlere Differenz in Bezug auf die maximale Punktzahl; siehe Tabelle 8, Seite 67). Die auf dem 5%-Niveau signifikanten Ergebnisse sind dick umrandet.

39 Berechnet mit dem Befehl: COMPUTE CDF.CHISQ(242.72,15).

		Verhalten (Taxonomien nach Bloom [Bl56])					
		Wissen	Verstehen	Anwendung	Analyse	Synthese	Evaluation
I	Konzepte						
n	Klasse & Objekt		B: -1,5%	C: 5,2%			
h	Beziehungen		E: 18%	F: 12%		H: 20%	I: 39%
a	Generalisierung		L: 16%	M: -3,6%		O: 2,6%	P: 7,4%
l	Methoden						
t	Klassenentwurf	Q: 48%	R: 13%				
e	Use Cases		U: 37%				
	UML	X: -9,9%		Y: -0,61%			

Tabelle 25. Vergleich der Lernziele in den beiden Gruppen[40]

Da auch hier nicht sicher ist, ob der t-Test angemessen ist, wird das Ergebnis mit dem verteilungsfreien Verfahren von Mann-Whitney überprüft. Dabei wird das Ergebnis des t-Tests vollständig bestätigt. Sowohl die signifikanten als auch die nicht signifikanten Ergebnisse sind identisch.
Die Teilnehmer der Umsetzungsgruppe erzielten beim Inhalt der *Beziehungen* auf allen Taxonomiestufen signifikant bessere Ergebnisse. Auch die höchste Taxonomiestufe, die *Evaluation* als der höchste kognitive Grad, ergibt dasselbe Bild: Die Methoden des objektorientierten Softwareentwurfs (Klassenentwurf, Use Cases) ergaben in der Umsetzungsgruppe ebenfalls signifikant bessere Ergebnisse.
Aus den letzten beiden Ergebnissen folgt, dass die Umsetzungsgruppe in der höchsten und anspruchsvollsten Taxonomiestufe der *Evaluation* und in den Inhalten *Methoden* besser ist. Die Schüler haben eher gelernt, komplexe Probleme in Modelle umzusetzen und sie so besser zu lösen.
In den Taxonomiestufen *Anwendung* und *Synthese* des Inhalts *Generalisierung* wird dagegen kein signifikanter Unterschied festgestellt.

Mit Hilfe einer Diskriminanzanalyse [Bo93] werden die Unterschiede in den Gruppen genauer untersucht. Dabei wird der Raum der abhängigen Variablen (hier: die Lernziele) über eine lineare Transformation so gedreht, dass die Umsetzungs- und die Kontrollgruppe optimal getrennt wird[41]. Da iterativ – Achse für Achse – vorgegangen wird, kommt der ersten neuen Achse (Diskriminanzfunktion 1) die Bedeutung der wichtigsten Trennung beider Gruppen zu.
Abbildung 31 zeigt die entstandene erste neue Achse (Diskriminanzfunktion 1) und die Verteilung der Messwerte abhängig von der Gruppenzugehörigkeit. Die Trennung der Gruppen ist deutlich zu erkennen.

40 Dick umrahmt sind die Lernziele, bei welchen signifikante Unterschiede festgestellt wurden (5%-Niveau). Die Prozentzahlen geben an, um wie viel Prozent die Umsetzungsgruppe im Mittel besser ist als die Kontrollgruppe, in Bezug zur maximalen zu erreichenden Punktzahl des Lernziels.

41 Das Verhältnis der Varianz zwischen den Gruppen (Hypothese) zur Varianz innerhalb der Gruppe (Fehler) wird durch diese Transformation maximiert.

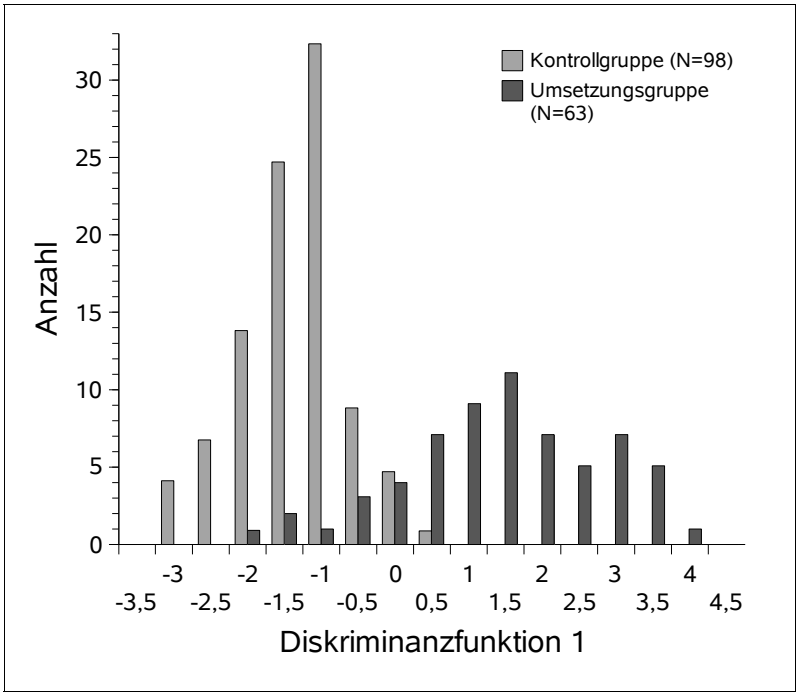


Abbildung 31. Trennung der beiden Gruppen durch die Diskriminanzfunktion

Die Diskriminanzanalyse ermöglicht dann eine Sortierung der abhängigen Variablen (Lernziele) nach ihrem trennenden Charakter. Die Korrelation der bisherigen abhängigen Variablen zur neuen Diskriminanzfunktion kann als eine Wichtigkeit für die Trennung der alten Variablen in Bezug auf die Gruppen aufgefasst werden. Je größer die Korrelation zwischen einer alten Variablen (hier: Lernziel) und der Transformierten, desto wichtiger ist diese alte Variable für die Trennung [Hu84].

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Lernziel	Q	U	I	H	F	L	E	R	P	X	C	M	O	B	Y
Korrelation	0,589	0,372	0,369	0,248	0,215	0,202	0,178	0,176	0,162	-0,106	0,088	-0,057	0,025	-0,025	-0,017

Tabelle 26. Die Korrelation zwischen den Diskriminanzvariablen (den Lernzielen) und der ersten standardisierten Diskriminanzfunktion. Die Variablen sind nach ihrer absoluten Korrelationsgröße sortiert.

Das Ergebnis dieser Berechnung ist Tabelle 26 zu entnehmen. Die Reihenfolge der Lernziele bestätigt Tabelle 25, die dort signifikanten 9 Lernziele sind auch hier die ersten 9 Lernziele, sortiert nach dem Korrelationskoeffizienten.

Sehr aufschlussreich ist auch die Durchführung einer Diskriminanzanalyse, bei welcher die insgesamt 7 teilnehmenden Schulklassen optimal getrennt werden sollen (siehe Abbildung 32). Werden die Messwerte der einzelnen Teilnehmer in diesen Raum transformiert, so zeigt sich, dass die Schüler der Umsetzungsgruppe (Klasse A, B, C) zum größten Teil einen Wert der Diskriminanzfunktion 1 größer 0, die Schüler der Kontrollgruppe (Klasse D, E, F1, F2) einen Wert von kleiner 0 besitzen.

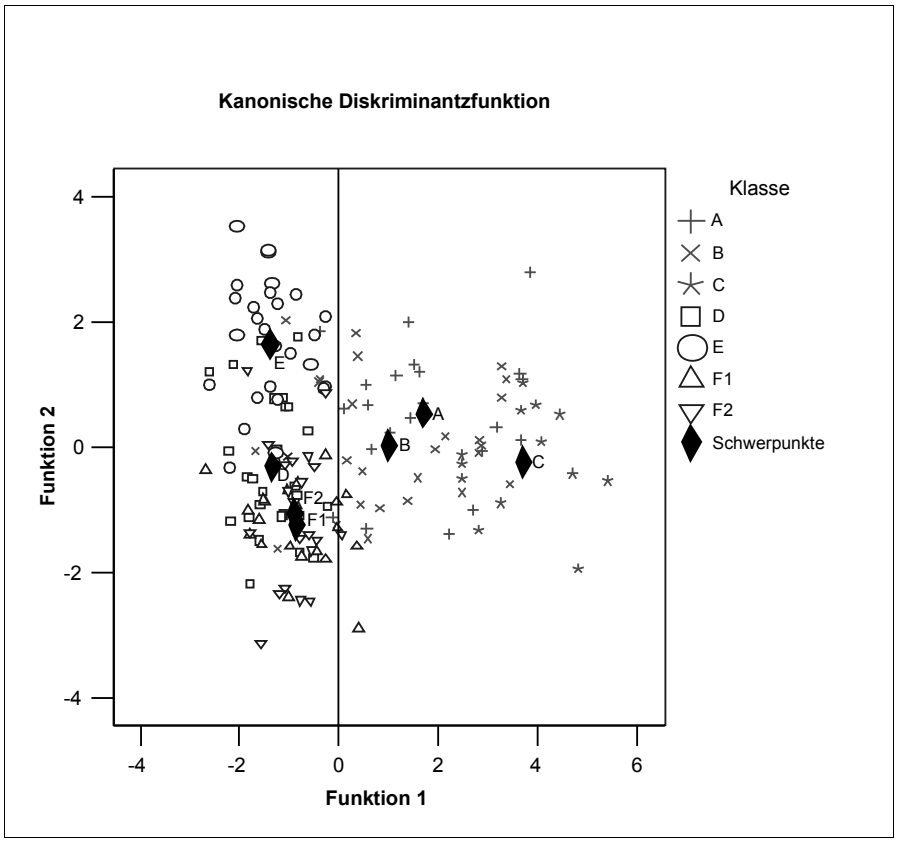


Abbildung 32. Diskriminanzanalyse zur optimalen Trennung der Schulklassen

Umsetzungsgruppe: A, B, C. Kontrollgruppe: D, E, F1, F2

Interessanterweise liegen die Schwerpunkte der Klasse F1 und F2 auch unter Einbeziehung der Diskriminanzfunktion 2, also der am zweitstärksten trennenden Achse, eng beieinander[42]. Da beide Schulklassen vom selben Lehrer unterrichtet wurden, könnte man hier auf eine Lehrerabhängigkeit schließen.

Zusammengefasst zeigt sich, dass die Umsetzungsgruppe auf der höchsten Lernzieltaxonomiestufe, der Evaluation, über alle getesteten Inhalte und im Bereich des Inhalts *Beziehungen* über alle getesteten Taxonomiestufen signifikant besser ist. Auch in den beiden methodischen Inhalten *Klassenentwurf* und *Use Cases* sind über alle getesteten Taxonomiestufen die Schüler der Umsetzungsgruppe signifikant besser.
Weiterhin zeigt die Diskriminanzanalyse zur Trennung der 7 Schulklassen, dass hier die Kontrollgruppe links und die Umsetzungsgruppe rechts auf der Achse der Diskriminanzfunktion 1 liegt, also wieder in Umsetzungs- und Kontrollgruppe getrennt wird, obwohl dieser Faktor bei der Berechnung nicht verwendet wurde.

4.3.5 Einfluss der Faktoren auf das Ergebnis

In diesem Kapitel soll untersucht werden, welche Faktoren das Ergebnis des Nachtests beeinflussen und an welcher Stelle dabei der Faktor *Zugehörigkeit zur Umsetzungs-/Kontrollgruppe* steht.

42 Die Diskriminanzfunktion 1 und 2 sind im statistischen Sinne orthogonal, d.h. unkorreliert.

4.3.5.1 Univariate Analyse

Angewandt wird zuerst eine univariate Varianzanalyse mit der abhängigen Variablen *Gesamtsumme des Nachtests*.

Für das univarianzanalytische Modell werden folgende Einflussfaktoren bzw. Kovariaten gewählt[43]:

- War der Schüler Mitglied der Umsetzungs- oder der Kontrollgruppe?
- In welcher Schulklasse war der Schüler?
  Über diesen Faktor soll die Abhängigkeit vom schulischen Umfeld in das Modell aufgenommen werden. Hierzu gehört u.a. die pädagogische und fachliche Kompetenz des Lehrers, die Klassengröße und die Anzahl der Rechner pro Schüler.
- Kann auf dem heimischen Rechner programmiert werden?
  Dazu wurden die Fragen des Fragebogens zur Erfassung des Schülerumfeldes bzgl. der Programmiersprache C++, Java und einer sonstigen Programmiersprache zusammengefasst.
- Beschäftigt sich der Schüler in der Freizeit mit dem Rechner?
- Programmiert der Schüler in der Freizeit auf dem Rechner?
- Spielt der Schüler in seiner Freizeit auf dem Rechner?
- Weiterhin, das Ergebnis des Intelligenztests als Kovariate,
- das Ergebnis des Vortests zum strukturierten Programmieren als Kovariate und
- die Einstellung zum Programmieren vor bzw. nach Vermittlung der Unterrichtseinheit als Kovariate.

Der Beruf des Vaters und der Mutter werden nicht in das Modell aufgenommen. Wie bereits in Kapitel 4.2 untersucht, besteht diesbezüglich kein Unterschied zwischen den beiden Gruppen. Sie dienen nur als Kontrollvariablen, um deren Einfluss auszuschließen. Durch die sehr große Anzahl der Ausprägungen würden recht kleine Gruppen entstehen, wobei einige Zellen nicht bzw. sehr schwach besetzt sind. Dies würde zu einer schlechten Statistik führen, d.h. einer Statistik mit großen Fehlern 1. Art (α-Fehler).
Auch das Vorhandensein eines heimischen Rechners und die installierten Programmierwerkzeuge (UML-Diagramme, Struktogramme) können als Faktoren nicht in das Modell aufgenommen werden, da zu wenig Schüler keinen Rechner bzw. Programmierwerkzeuge installiert haben und dadurch wiederum eine schlechte Statistik entstehen würde.

Wird im Modell sowohl die Schulklassenzugehörigkeit als auch die Zugehörigkeit zur Umsetzungs-/Kontrollgruppe verwendet, so kann SPSS für diesen Parameter keine Werte (z.B. α-Fehler, Varianzaufklärung) ermitteln.
Dies ist eine Folge der direkten Abhängigkeit dieser beiden Faktoren (Korrelation r=1). Der Freiheitsgrad für den Faktor *Umsetzungs-/Kontrollgruppe* beträgt $df_{treat}=0$, wenn die Schulklasse gegeben ist, da durch die Zugehörigkeit zur Schulklasse die Gruppe bereits festliegt und somit keine Freiheit besteht. Somit ist hier die Varianz [Bo93, S. 228ff]

43 Kategoriale Variablen werden hier als Faktoren bezeichnet. Sie trennen verschiedene Kategorien und sind entweder nominal oder ordinal skaliert, so dass höchstens eine Rangfolge angegeben werden kann. Kovariaten sind im Gegensatz hierzu intervallskaliert. Dadurch können Korrelationen mit der abhängigen intervallskalierten Variablen ermittelt werden.

$$\sigma^2_{treat} = \frac{QS_{treat}}{df_{treat}} \tag{4.19}$$

mit

$$QS_{treat} = n \cdot \sum_i (\bar{A}_i - \bar{G})^2 \tag{4.20}$$

als der Treatment-Quadratsumme und $\bar{A}_i$ als dem Mittelwert innerhalb einer Ausprägung des Treatment-Faktors mit *i* Stufen, $\bar{G}$ als dem Mittelwert über alle Messwerte und *n* der Anzahl der Messwerte, nicht berechenbar, da eine Division durch 0 vorliegen würde. Die Angabe einer Varianz ist bei einer Korrelation von r=1 nicht sinnvoll.
Da sich nun die F-Verteilte Testgröße nach

$$F = \hat{\sigma}^2_{treat} / \hat{\sigma}^2_{Fehler} \quad , \tag{4.21}$$

mit $\hat{\sigma}^2$ der Varianz des Treatments (siehe Gleichung 4.19) bzw. des Fehlers berechnet, ist auch dieser Wert nicht zu ermitteln, so dass kein α-Fehler angeben werden kann. Dies ist auch sinnlos, da aus dem Faktor *Schulklasse* mit insgesamt 7 Stufen (7 Schulklassen), direkt der Wert des Faktors *Umsetzungs-/Kontrollgruppe* errechnet werden kann (nicht jedoch umgekehrt).
Aus den genannten Gründen werden zwei Analysen durchgeführt: Einmal unter Verwendung der Zugehörigkeit zur Umsetzungs-/Kontrollgruppe, das andere Mal unter Verwendung der Zugehörigkeit zur Schulklasse.

Um die Größe des Einflusses eines Parameters anzugeben, wird die Varianzaufklärung η^2 verwendet [Bo93, S. 228ff und S. 274ff]. Dazu wird die Quadratsumme des Treatments QS_{treat} (siehe Gleichung 4.20) in das Verhältnis zur gesamten Quadratsumme

$$QS_{tot} = \sum_i \sum_m (x_{mi} - \bar{G})^2 \tag{4.22}$$

mit x_{mi}, des m. Messwerts des i. Treatments, gesetzt und dieser Wert in Prozent angegeben, also

$$\eta^2_A = \frac{QS_{treat,A}}{QS_{tot}} \quad . \tag{4.23}$$

Sie gibt an, welchen prozentualen Anteil das Treatment A (z.B. der Intelligenzquotient IQ) an der gesamten Varianz besitzt und ist somit ein Wert für den Einfluss des Treatments auf das Ergebnis. Es gilt

$$\eta^2_{ges} = \eta^2_A + \eta^2_B + \ldots + \eta^2_{AxB} + \ldots + \eta^2_{fehler} = 100\,\% \quad , \tag{4.24}$$

wobei η^2_{AxB} der Einfluss der Interaktion[44] des Treatments A mit Treatment B auf die gesamte Varianz und η^2_{fehler} den Anteil unbekannter bzw. nicht erhobener Treatments (z.B. der Einfluss des Wetters) darstellt. Anteile in dieser Summe, welche nicht signifikant sind, hier also mit einer Signifikanz $\alpha > 5\,\%$, werden mit dem Fehler zusammengefasst und als unaufgeklärte Varianz bezeichnet.

44 Dabei wird berücksichtigt, dass z.B. das Treatment A (z.B. das Ergebnis der Struktogrammaufgabe) nur bei großem Treatment B (z.B. dem IQ) einen Einfluss besitzt. Bei mehreren Treatments entstehen dabei auch größere Interaktionen wie z.B. AxBxC.

Das Ergebnis unter Verwendung der Zugehörigkeit zur Umsetzung-/Kontrollgruppe zeigt Abbildung 33. Interaktionen zwischen Faktoren und/oder Kovariaten sind nicht angegeben, da diese alle auf dem 5%-Niveau nicht signifikant sind.

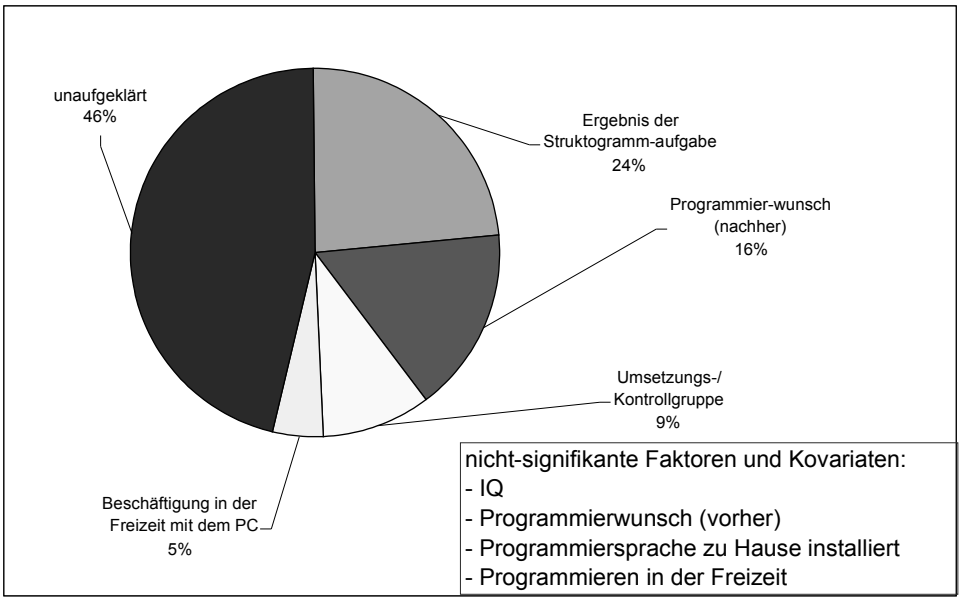


Abbildung 33. Die Varianzaufklärung η^2 der Faktoren zum Gesamtergebnis des Nachtests unter Berücksichtigung der Zugehörigkeit zur Umsetzungs-/Kontrollgruppe (Signifikanzniveau 5%)

Durch dieses Modell wird 54% der Varianz erklärt, d.h. die im Modell aufgenommenen und signifikanten Faktoren begründen 54% der gesamten Varianz. Die Zugehörigkeit zur Umsetzungs-/Kontrollgruppe ergibt einen Anteil von 9% und ist der drittwichtigste Faktor. Der wichtigste Faktor ist jedoch die Fähigkeit strukturiert zu programmieren (24%), gefolgt vom *Programmierwunsch*, welcher nach Ende der Unterrichtseinheit ermittelt wurde (16%).

Abbildung 34 zeigt die Analyse unter Verwendung der Zugehörigkeit zur Schulklasse. Es zeigt sich, dass der Einfluss der Schulklasse (20%) ca. doppelt so groß ist wie der Einfluss der Mitgliedschaft in der Umsetzungs-/Kontrollgruppe wie in Abbildung 33. Einziger weiterer signifikanter Faktor ist der *Programmierwunsch* nach Vermittlung der Einheit (7%). Der Anteil der unaufgeklärten Varianz steigt auf 73% stark an.

Da laut Bortz [Bo93, S. 263]

> *„bei ungleichgroßen Stichproben und heterogenen Varianzen [...] die Gültigkeit des F-Tests vor allem bei kleineren Stichprobenumfängen erheblich gefährdet“*

ist, wird auch die Heterogenität der Varianzen untersucht, da die Stichproben ungleich groß und eher klein sind[45].

45 Zum Beispiel Größe der Schulklassen von 14 bis 30 Schülern (siehe Tabelle 11, S. 75)

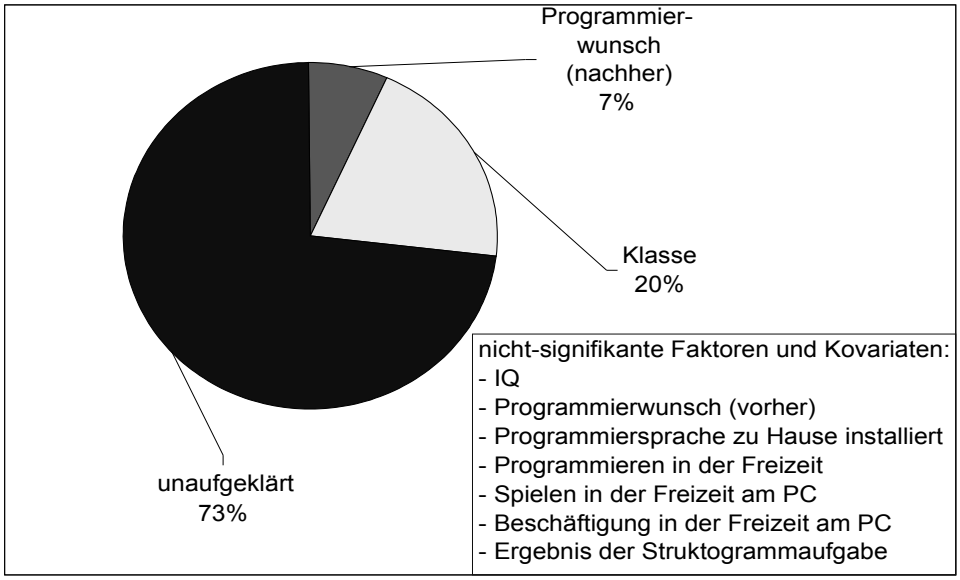


Abbildung 34. Die Varianzaufklärung η^2 der Faktoren zum Gesamtergebnis des Nachtests unter Berücksichtigung der Zugehörigkeit zur Klasse (Signifikanzniveau 5%)

Dazu eignet sich der Levene-Test [Po02, S. 6-50], welcher die Nullhypothese testet, dass die Varianzen in allen Gruppen homogen sind. Die Homogenität ist gegeben, wenn die Signifikanz > 20% ist.
Dieser Test ergibt in der Untersuchung mit dem Faktor *Umsetzungs-/Kontrollgruppe* den Wert 13,6%, für die Untersuchung mit dem Faktor *Schulklasse* den Wert 0,8%. Dies bedeutet, dass vor allem bei der zweiten Untersuchung die Varianzen stark heterogen sind und deswegen nach Bortz die Gültigkeit des F-Tests erheblich gefährdet ist.

4.3.5.2 Multivariate Analyse

Wie in vorangegangenen Kapiteln bereits erwähnt, wird bei der multivariaten Analyse des Faktors *Gesamtsumme des Nachtests* implizit eine Gewichtung der Lernziele vorgenommen. Deswegen ist der Ansatz einer multivariaten Analyse mit einem n-dimensionalen Ergebnisraum der bessere Weg.
Auch hier soll die Frage geklärt werden, welche Faktoren das Ergebnis signifikant beeinflussen und an welcher Stelle dabei der Faktor *Zugehörigkeit zur Umsetzungs-/Kontrollgruppe* steht. Das Modell wird im Vergleich zur univariaten Analyse nicht geändert. Interaktionen zwischen Faktoren und/oder Kovariaten werden ebenfalls nicht betrachtet. Auch hier ergibt sich wieder das obige Problem, dass die Faktoren *Mitglied in der Umsetzungs-/Kontrollgruppe* und *Schulklasse* nicht zusammen verwendet werden können. Deswegen werden beide wiederum getrennt betrachtet.

	Umsetzungs-/Kontrollgruppe		Schulklasse	
	normalvert.	vert. frei	normalvert.	vert. frei
Umsetzungs-/Kontrollgruppe	S	S	/	/
Schulklasse	/	/	S	S
Prog. zu Hause vorhanden	S	S	-	S
Freizeitbesch. mit dem Rechner	-	-	-	-
Prog. in der Freizeit	S	S	S	S
Spielen in der Freizeit mit dem Rechner	-	-	-	-
IQ	-	-	-	-
Erg. der Struktogrammaufgabe	S	S	-	-
Programmierwunsch (vorher)	-	-	-	S
Programmierwunsch (nachher)	-	-	-	-

Tabelle 27. Das Ergebnis des multivariaten Vergleichs (unter Annahme der Normalverteilung bzw. ohne), abhängig von den Faktoren[46]

Da auch hier die Voraussetzung der n-dimensionalen Normalverteilung der abhängigen Variablen (Ergebnisse innerhalb der einzelnen Lernziele) fragwürdig ist, wird neben der Varianzanalyse auch noch das oben bereits erwähnte verteilungsfreie Verfahren nach Zwick [Zw85] verwendet. Alle Ergebnisse sind Tabelle 27 zu entnehmen.

Der Faktor *Umsetzungs-/Kontrollgruppe* bzw. *Schulklasse* spielt auch bei der multivariaten Analyse sowohl unter der Voraussetzung der Normalverteilung der abhängigen Variablen (hier: der Lernziele) als auch ohne diese Voraussetzung eine signifikante Rolle. Unter Verwendung des Faktors *Umsetzungs-/Kontrollgruppe* sind des Weiteren die Faktoren *Programmiersprache zu Hause vorhanden*, *Programmieren in der Freizeit* und *Ergebnis der Struktogrammaufgabe* signifikant, d.h. sie beeinflussen das Ergebnis. Dies bedeutet, dass das Vorhandensein einer Programmiersprache zu Hause und das Damit-Arbeiten (Programmieren in der Freizeit) das Ergebnis des Nachtests signifikant beeinflusst. Ebenfalls besteht ein signifikanter Zusammenhang zum Ergebnis der Struktogrammaufgabe.
Unter Verwendung des Faktors *Schulklasse* ergibt sich ein etwas anderes Bild. Der Faktor *Programmieren in der Freizeit* bleibt signifikant, der Faktor *Programmiersprache zu Hause vorhanden* ist nur bei der Untersuchung ohne die Voraussetzung der Normalverteilung signifikant. Zusätzlich beeinflusst der Faktor *Programmierwunsch (vorher)*, ebenfalls nur unter der Annahme der nicht Normalverteilung der Lernziele, das Ergebnis des Nachtests.

Insgesamt ist also zu sagen, dass einerseits die Gruppenzugehörigkeit bzw. die Schulklassenzugehörigkeit und das Programmieren in der Freizeit das Ergebnis des Nachtests eindeu-

46 S = signifikante Beeinflussung auf dem 5%-Niveau. / = nicht im Modell enthalten. Signifikante Beeinflussungen bei Interaktion zwischen Faktoren und/oder Kovariaten sind nicht angegeben.

tig beeinflusst. Das Programmieren in der Freizeit hat das Ergebnis des Nachtests also signifikant, genauso wie die Zugehörigkeit zur Umsetzungsgruppe verbessert.

4.3.6 Einschätzungen der Lehrer und Schüler

Nach Vermittlung der gesamten Lehrplaneinheit wurden die Lehrer und Schüler zu ihrem subjektiven Empfinden bezüglich des Unterrichts gefragt.

4.3.6.1 Lehrerbefragung

Die Ergebnisse der Lehrerbefragung sind Tabelle 28 zu entnehmen. Signifikante statistische Aussagen sind nicht möglich, da die beiden Gruppen zu klein sind.
Im Bereich der fachlichen und pädagogischen Kenntnisse schätzten sich die Lehrer der Umsetzungsgruppe im Mittel wiederum wie bei der Befragung vor Vermittlung der Lehrplaneinheit (siehe Tabelle 15 auf S. 78) besser ein als die Lehrer der Kontrollgruppe.

	Umsetzungsgruppe		Kontrollgruppe	
	Mittelwert	Einzelwerte	Mittelwert	Einzelwerte
Fachlich				
Programmierung	8,3	8; 8; 9	7,0	8; 4; 9
Objektorientierung	8,0	10; 6; 8	6,0	6; 4; 8
Pädagogisch				
Programmierung	7,0	7; 9; 5	6,3	9; 3; 7
Objektorientierung	7,0	8; 5; 8	6,0	9; 3; 6
Lehrerspaß	8,0	9; 8; 7	7,3	9; 7; 6
Schülermotivation	6,0	8; 6; 4	5,7	7; 5; 5
Abivorbereitung	8,0	9; 7; 8	4,5	- ; 3; 6
Studienvorbereitung	7,7	9; 6; 8	6,3	8; - ; 4,5

Tabelle 28. Die subjektive Einschätzung der Lehrer nach Vermittlung der gesamten Lehrplaneinheit

Auch der Mittelwert der Frage zum *Lehrerspaß* fällt in der Umsetzungsgruppe etwas besser aus. Der Mittelwert der Schülermotivation aus Sicht der Lehrer unterscheidet sich hingegen kaum. Demgegenüber haben die Lehrer der Umsetzungsgruppe das Gefühl, dass die Schüler besser für das Abitur und ein Studium vorbereitet sind.

Die Kommentare der teilnehmenden Lehrer der Umsetzungsgruppe nach Ende der Lehrplaneinheit zeigen, dass die Übungsphasen und praktischen Phasen sehr zeitintensiv sind und kaum in das Zeitraster passen. Allerdings wird auch betont, dass diese Phasen wichtig sind. Weiterhin wurde von einem Lehrer bemängelt, dass das gewählte Beispiel der Kobolde nicht mehr altersgerecht ist. Dieser Lehrer wird das Konzept weiter umsetzten, allerdings anstatt der Kobolde mit Lokomotiven arbeiten.
Auch sonst gaben alle Lehrer an, das Konzept weiter einsetzen und weiterentwickeln zu wollen.

4.3.6.2 Schülerbefragung

Die Auswertung der anonymen Schülerfragebögen zur subjektiven Einschätzung nach Vermittlung der Einheit ergibt kaum Unterschiede in den beiden Gruppen (siehe Abbildung 35). Ein t-Test ergibt nur für das Item *das Thema fand ich spannend* (Fragebogen siehe Anhang C.5) eine auf dem 5%-Niveau besseren Wert für die Umsetzungsgruppe. Ein durchgeführter Mann-Whitney-Test[47] kann dies, allerdings nur mit einer Restfehlerwahrscheinlichkeit von α=5,3%, bestätigen.

Die Ergebnisse liegen alle im mittleren Feld. Hinsichtlich der Einschätzung der Schüler bezüglich des *Berufswunsches in der Informatik*, *besser programmieren zu können*, *Spaß zu haben* und das *Thema spannend zu finden* werden keine Unterschiede zwischen den beiden Gruppen festgestellt. Auch hatten die Schüler in beiden Gruppen die subjektive Einschätzung, dass dem Lehrer das Thema mittelmäßig *Spaß machte*.

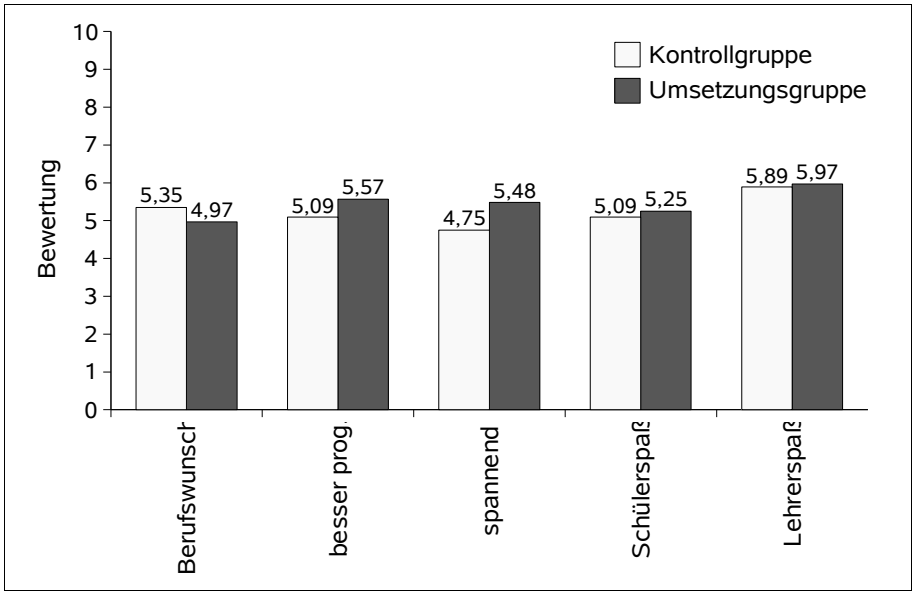


Abbildung 35. Die Mittelwerte der Schülerbefragung nach Durchführung der Unterrichtseinheit

In Abbildungen 36 und 37 ist ein Vergleich der Inhalte zu finden, welche aus Sicht der Schüler am besten bzw. am schlechtesten verstanden wurden (Fragebogen siehe Anhang C.5).

Wurden von einem Schüler 2 Bereiche angegeben, so werden beide in diese Kategorie aufgenommen, so dass die maximale Anzahl der Nennungen (196) der doppelten Anzahl der Schüler (98) entspricht. In der Abbildung sind die Angaben auf die maximale Anzahl der Nennungen (196) normiert.

Ein signifikanter Unterschied auf dem 5%-Niveau zwischen beiden Gruppen wird mit Kontingenztafeln nach Borz [Bo00, S.158f] nachgewiesen.

In der subjektiven Selbsteinschätzung gaben die Schüler der Umsetzungsgruppe die Inhalte *Beziehungen, Rollen und Kardinalitäten* und *Entwurf des Klassenmodells* signifikant häufiger als am besten verstandenen Inhalt an. Demgegenüber gaben in der Kontrollgruppe die Schüler die Inhalte *Generalisierung, Vererbung* und *UML* signifikant häufiger an.

47 Der Mann-Whitney-Test geht nicht von der Annahme der Normalverteilung der Daten aus.

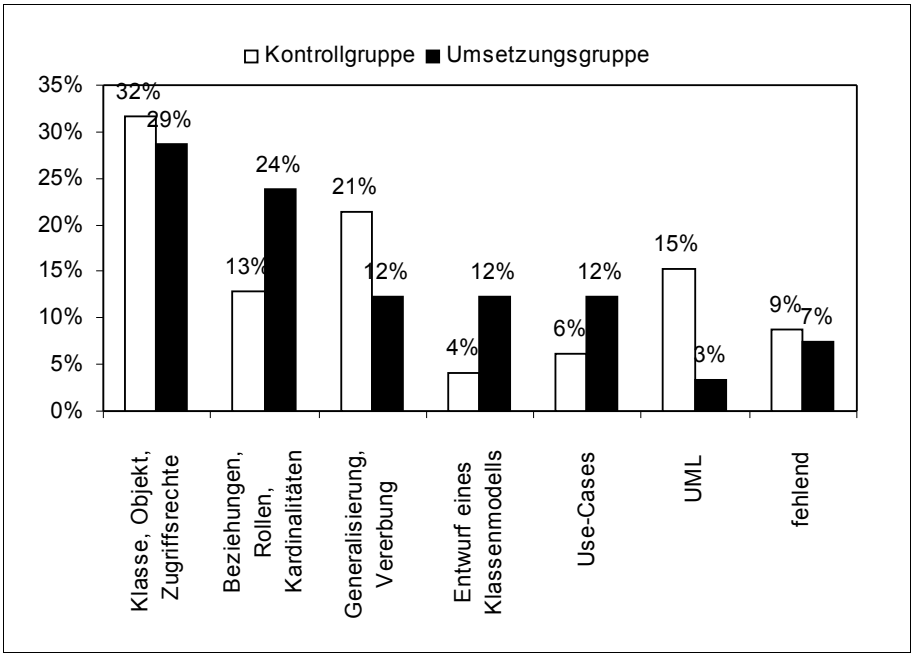


Abbildung 36. Die am besten verstandenen Inhalte in der Umsetzungs- und Kontrollgruppe

Die Schüler der Umsetzungsgruppe meinten also, eher die Inhalte *Beziehungen, Rollen und Kardinalitäten* und *Entwurf des Klassenmodells* und die Schüler der Umsetzungsgruppe die Inhalte *Generalisierung, Vererbung* und *UML* am besten verstanden zu haben.

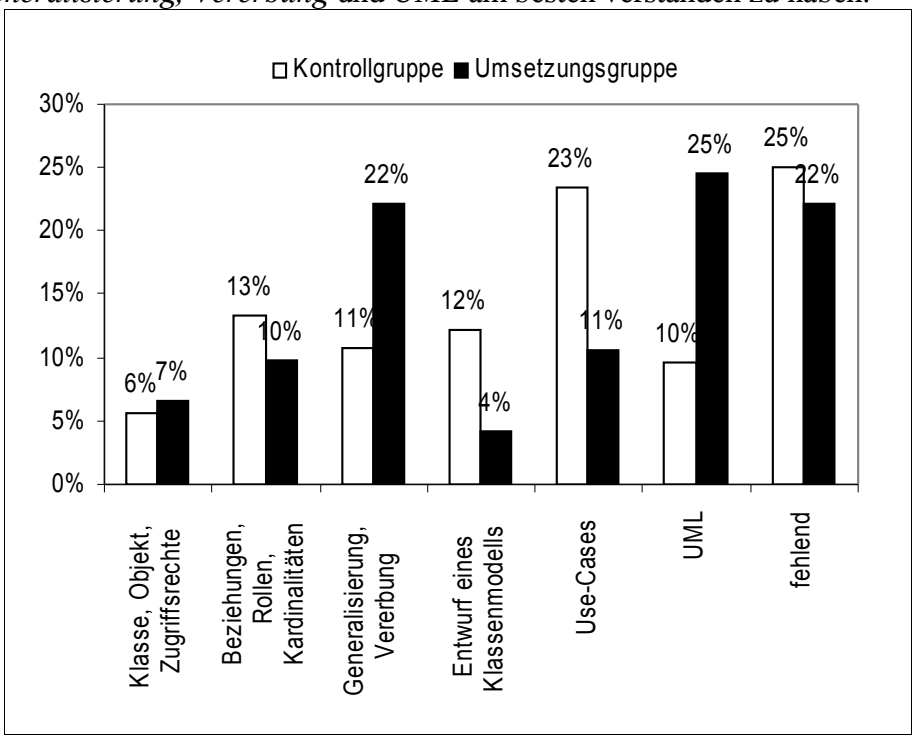


Abbildung 37. Die am schlechtesten verstandenen Inhalte in der Umsetzungs- und Kontrollgruppe

Für die am schlechtesten verstandenen Inhalte gilt nahezu die umgekehrte Sicht. Hier gaben die Schüler der Umsetzungsgruppe die Inhalte *Generalisierung, Vererbung* und *UML* signifikant häufiger als am schlechtesten verstandene Inhalte an. Demgegenüber gaben in der Kontrollgruppe die Schüler die Inhalte *Entwurf des Klassenmodells* und *Use Cases* signifikant häufiger an.
Die Schüler der Umsetzungsgruppe meinten also, eher die Inhalte *Generalisierung, Vererbung* und *UML* und die Schüler der Kontrollgruppe die Inhalte *Entwurf des Klassenmodells* und *Use Cases* am schlechtesten verstanden zu haben.

		N	Mittel	Std. Abweichung
Umsetzungsgruppe	vorher	47	6,38	2,481
	nachher	46	5,85	2,641
Kontrollgruppe	vorher	92	6,80	2,175
	nachher	90	6,04	2,060

Tabelle 29. Der Wunsch zu programmieren vor und nach der Vermittlung der Einheit in den beiden Gruppen

Der Wunsch zu programmieren wurde vor und nach der Einheit erfragt (siehe Tabelle 29). Zwischen den beiden Gruppen besteht sowohl vor als auch nach Vermittlung der Einheit kein signifikanter Unterschied.
In der Kontrollgruppe verringerte sich der Programmierwunsch signifikant zwischen Beginn und Ende der Unterrichtseinheit. In der Umsetzungsgruppe verschlechterte sich zwar der Wert, allerdings ohne Signifikanz.

4.3.7 Unterrichtsbesuch

Mit den beiden Unterrichtsbesuchen[48] sollte einerseits die persönliche Umsetzung des Lehrers, andererseits ein Eindruck von der Meinung der Schüler zum Konzept gefunden werden.
Beim Besuch wurde als Kritik von den Schülern beide Male der recht anstrengende Einstieg in das Programm *Willis Welt* (siehe Kapitel 3.7.2) geäußert. Es handelt sich zwar um ein recht einfaches Programm, um jedoch damit arbeiten und Änderungen bzw. Erweiterungen vornehmen zu können, ist ein tieferes Verständnis sehr wichtig. Dazu sind längere Erklärungsabschnitte notwendig, was von den Schülern als recht anstrengend empfunden wurde. Da aber danach dasselbe Programm immer weiter verwendet wird, stellt dies nur beim Einstieg ein Problem dar.

Schule C wurde während der Unterrichtseinheit *Übergang zum Klassendiagramm mit Hilfe des Sequenz- und Kollaborationsdiagramms* (siehe Anhang A.2.6) besucht. Ziel dieser Einheit ist es, den Schülern die Bedeutung und Hilfe, welches das Sequenzdiagramm bietet, zu zeigen. Damit lassen sich durch sequentielles Durchdenken des Ablaufs Methoden in den unterschiedlichen Klassen finden und der zeitliche Ablauf übersichtlich darstellen. In der

48 Es wurden nur zwei Schulen besucht, da die dritte Klasse der Umsetzungsgruppe der Ersteller des Konzepts ist.

Schulklasse entstand dazu eine kleine Diskussion, welche Methode zu welcher Klasse gehört. Dies ist eine zentrale Frage beim objektorientierten Softwareentwurf, weswegen diese Diskussionen immer wieder anzustreben sind, um so den *Informationsexperten* (siehe Kapitel 3.4) zu finden.
Es zeigt sich, dass hier der Lehrer ein gutes Fachwissen besitzen muss, um so adäquat auf die Fragen und Anregungen der Schüler eingehen und das Gespräch so besser moderieren und steuern zu können. Nur so kann er in der Diskussion immer wieder auf die zentralen Inhalte zurückführen.
Am Ende der Doppelstunde wurde dann noch auf die Bedeutung des 2-Schichtenmodells eingegangen (View-Schicht, Control-Schicht, S. 137f). Dies ist zwar im Konzept erst später vorgesehen, kann aber an den entsprechenden Stellen auch vorher eingebracht werden.

Schule B wurde zur Besprechung der Wettkampf-Aufgabe besucht (siehe Anhang B.3). Dabei zeigte sich die Schwäche der Aufgabe, die Kardinalitäten sind zum Teil nicht eindeutig zu finden, da eine genauere Erklärung im Text fehlt. Hier zeigt sich ein allgemeines Problem bei dieser Art von Aufgaben: Einerseits sollten die Probleme nicht zu trivial sein, andererseits ergibt sich bei angemessenem Umfang der Lösung das Problem, dass dann die Aufgabe selbst recht umfangreich wird, um Zweideutigkeiten zu vermeiden.
Im realen Softwareentwurf ist dies eine Aufgabe des Analysten, diese zweideutigen Bereiche zu finden und mit dem Kunden zu klären. Die UML-Analysediagramme geben dann die gefundenen Verdeutlichungen wieder, welche aus einer Diskussion zwischen Analysten und Kunde entstanden.
Im Unterricht sollte diese Diskussion ebenfalls nachvollzogen werden. Allerdings exisiert bei den Schülern eine recht hohe Erwartung, dass die Aufgabe eindeutig gestellt und eindeutig gelöst werden kann. In Bezug auf Klassenarbeiten und das Abitur ist dies aus Sicht der Schüler auch verständlich. Allerdings versperrt dies das Verständnis, dass auch in der Realität oftmals erst um die Eindeutigkeit gerungen werden muss.
Wie bereits bei der anderen Schule erwähnt, zeigt sich wiederum, dass der Lehrer solides Fachwissen benötigt, um diese Diskussionen angemessen zu moderieren. Er muss auf den genannten Zwiespalt hinweisen und von den Schülern beide Fähigkeiten verlangen, d.h. mit eindeutig gestellten Aufgaben zur Lösung von Schulklassen- und Abituraufgaben, und mit mehrdeutig gestellten, um die realen Probleme im Alltag zu meistern, umgehen können.

Insgesamt haben die beiden Besuche gezeigt, dass das Konzept richtig umgesetzt wird. Die Intention, über laufend stattfindende Diskussionen und Gespräche die Fähigkeiten der Schüler im Softwareentwurf zu verbessern, wurde umgesetzt. Das Konzept bietet hierzu vielfältige Anregungen.
Andererseits zeigt sich auch die Problematik des komplexen und abstrakten Inhalts der verlangten Lernziele. Bereits kleine reale Problemstellungen ergeben recht komplexe Modelle.

4.3.8 Informatische Vorerfahrungen

In der Aufgabe zur Ermittlung der informatischen Vorerfahrung (siehe Aufgabe 1, Anhang C.6) erreichten die Schüler im Schnitt 49,3% der möglichen Punkte.

Ein Vergleich der beiden Gruppen zeigt Abbildung 38. Dabei werden die Mittelwerte auf den jeweiligen maximal erreichbaren Wert normiert. Neben dem Gesamtergebnis der Auf-

gabe werden auch die einzelnen Bewertungspunkte, wie sie in Kapitel 4.1.5 beschrieben sind, verglichen. Ein t-Test ergibt nur für den Bewertungspunkt *Auswahl* ein signifikant besseres Abschneiden der Umsetzungsgruppe. Dies kann jedoch vom verteilungsfreien Mann-Whitney-Test nicht bestätigt werden. Dort gibt es für keinen Bewertungspunkt signifikante Unterschiede zwischen den beiden Gruppen.
Insgesamt wird deswegen die Nullhypothese, dass die beiden Gruppen unterschiedliche informatische Vorerfahrungen haben, verworfen.

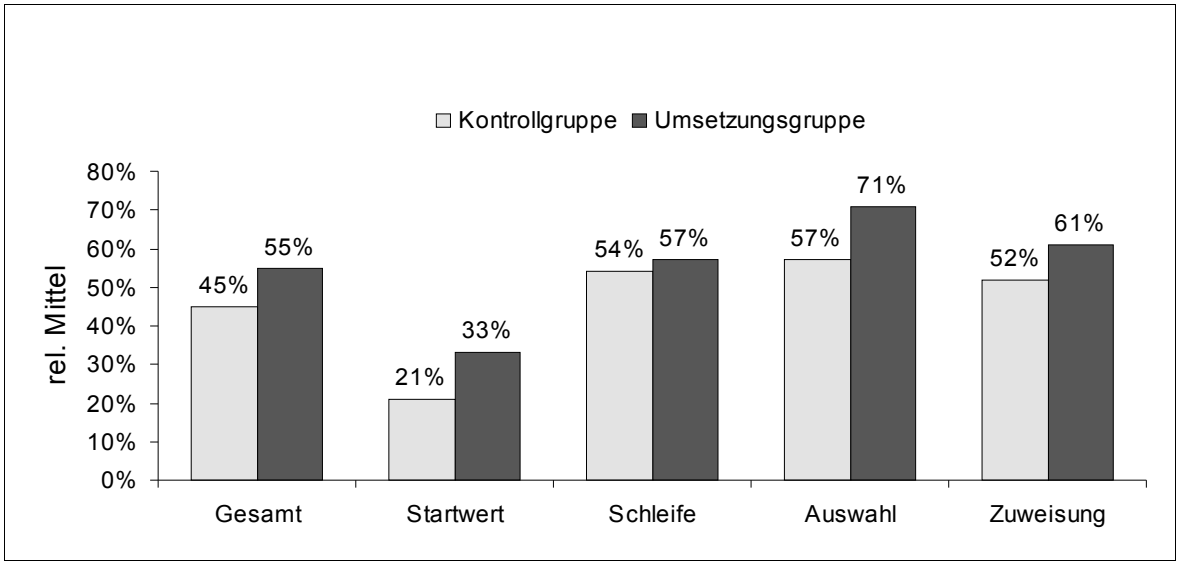


Abbildung 38. Vergleich zwischen Umsetzungs- und Kontrollgruppe der Aufgabe zur informatischen Vorerfahrung

Zusätzlich wird multivariat der Einfluss verschiedener Faktoren geprüft (Verfahren wie in Kapitel 4.3.5.2), um evtl. Abhängigkeiten bei der Vorerfahrung zu erkennen. Dabei werden die 4 Bewertungskategorien als abhängige Variablen verwendet und folgendes multivariates Modell gewählt:

- Zugehörigkeit zur Umsetzungs-/Kontrollgruppe als Faktor,
- Zugehörigkeit zur Klasse als Faktor, um Abhängigkeiten von den Randbedingungen in den Klassen (Lehrer, Klassengröße, Stundenanzahl, ...) zu ermitteln,
- der Wunsch zu programmieren vor Vermittlung der Unterrichtseinheit und
- der Intelligenzquotient, jeweils als Kovariate.

Auch hier ergibt sich die bereits in Kapitel 4.3.5.1 erwähnte Problematik der gleichzeitigen Verwendung der Faktoren *Umsetzung-/Kontrollgruppe* und *Schulklasse* in einem Modell, so dass auch hier getrennt untersucht wurde.

Zuerst wird der Faktor *Umsetzungs-/Kontrollgruppe* verwendet. Das Ergebnis des multivariaten Tests ergibt nur eine signifikante Abhängigkeit vom Faktor *Programmierwunsch vor Vermittlung der Unterrichtseinheit*. Allerdings ergibt ein multivariater Test ohne die Voraussetzung der Normalverteilung der abhängigen Variablen (nach Zwick [Zw85]; siehe Kapitel 4.3.3.2) keine signifikante Abhängigkeiten von den Variablen des Modells, so dass die Abhängigkeit vom *Programmierwunsch vor Vermittlung der Unterrichtseinheit* nicht bestätigt wird.

Verwendet man statt des Faktors *Umsetzungs-/Kontrollguppe* den Faktor *Schulklasse*, so ergibt sich sowohl unter Berücksichtigung der Normalverteilung der Bewertungspunkte als auch ohne diese Voraussetzung jeweils eine signifikante Abhängigkeit von der Schulklasse. Die anderen Parameter ergeben kein signifikantes Ergebnis.

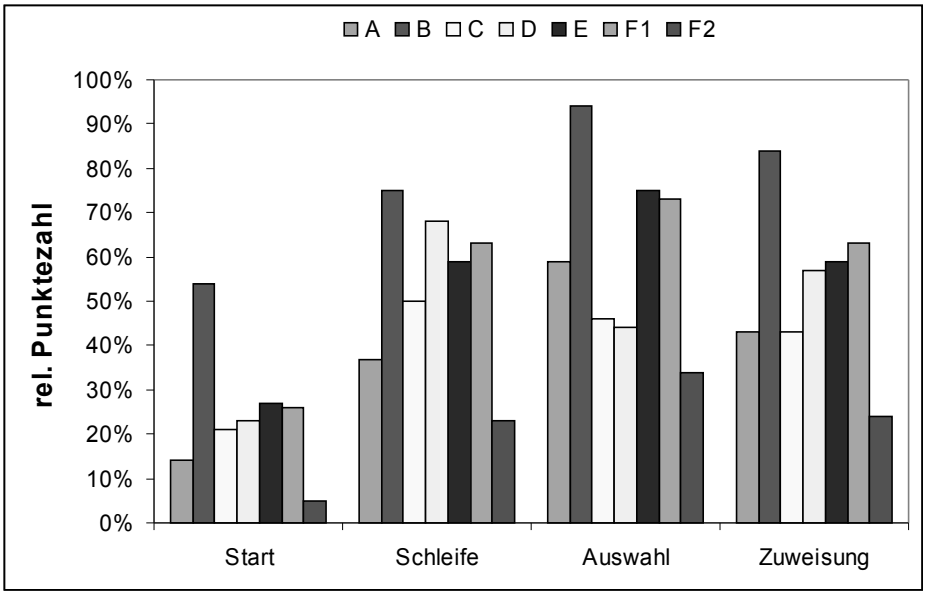


Abbildung 39. Der Mittelwert der Ergebnisse zum Finden des Algorithmus abhängig von der Schulklasse und den Bewertungspunkten

Abbildung 39 zeigt die Mittelwerte normiert auf die maximale Punktzahl der einzelnen Bewertungspunkte für alle Schulklassen. Gut zu erkennen ist das stark von der Schulklasse abhängige Ergebnis für die einzelnen Bewertungspunkte.

5 Diskussion und Ausblick

5.1 Diskussion der Evaluationsergebnisse

Es wurde ein Konzept zur Vermittlung des objektorientierten Softwareentwurfs entworfen und evaluiert. Mit Hilfe der Evaluation sollte geklärt werden, ob das entworfene Konzept für die Sekundarstufe II geeignet ist, die Lernziele erreicht werden und die Schüler im Vergleich zu einer Kontrollgruppe in Bezug auf diese Lernziele besser abschneiden.
Nachfolgend sollen die Ergebnisse der Evaluation diskutiert werden.

Rahmenbedingungen

Als Erstes sollen die Ergebnisse bzgl. der Ermittlung der Rahmenbedingungen erörtert werden (siehe Kapitel 4.2).
Dabei fällt der hohe Anteil an der Verfügbarkeit eines heimischen Rechners bei allen Schülern auf. In der Umsetzungsgruppe stand zu 100% ein heimischer Rechner zur Verfügung. Auch in der Kontrollgruppe besitzen 89% einen heimischen Rechner zum Üben. Die schulische Programmiersprache war in der Umsetzungsgruppe signifikant öfter installiert als in der Kontrollgruppe. Trotzdem wird in der Umsetzungsgruppe in der Freizeit signifikant weniger programmiert als in der Kontrollgruppe. Die Verfügbarkeit von Ressourcen ist also keine Gewähr für ihre Verwendung.
Das Vorhandensein eines heimischen Rechners und der schulischen Programmiersprache führt nicht unbedingt zu einer Beschäftigung mit informatischen Themen. Es zeigt sich, dass der Rechner hauptsächlich als Spielgerät verwendet wird und weniger zum Programmieren. Dies ist deswegen verwunderlich, da die Schüler sich mit der Wahl für das informationstechnische Gymnasium für eine Schulart mit vielen informatischen Inhalten entschieden haben. Es scheint so, dass der Unterschied zwischen Rechnerbenutzung und Informatik als Fachdisziplin bei der Entscheidung nicht klar oder lediglich der Wunsch vorhanden war, sich dieser Fachdisziplin zu nähern.
An dieser Stelle wäre eine Untersuchung interessant, die klärt, welche Kriterien bei der Wahl des informationstechnischen Gymnasiums eine Rolle spielen. In diesem Zusammenhang müsste man auch das Bild der Informatik in den Köpfen der Schüler untersuchen.

Die Inhalte des objektorientierten Softwareentwurfs haben sich die weitaus meisten Kollegen im Selbststudium erarbeitet. Dies bedeutet einen großen Aufwand und ist im Vergleich zu einer angeleiteten Fortbildung nicht so effektiv, da die Schwerpunkte mit viel mehr Aufwand erarbeitet werden müssen. Es zeigt sich auch, dass nur geringe Erfahrung bei den Lehrern für diesen Bereich vorhanden ist und sie für jede Hilfe dankbar sind.
Daraus ergibt sich automatisch die Forderung, dass informatische Inhalte und vor allem hier der objektorientierte Softwareentwurf verstärkt in Fortbildungen vermittelt werden müssen. Dies bezieht sich nicht nur auf den pädagogischen, sondern auch auf den fachwissenschaftlichen Bereich.

Die materielle Ausstattung scheint in diesem Bereich hingegen recht gut zu sein. Für das Erlernen des Codierens, wie es im Fach Computertechnik vorgesehen ist, ist ein eigener Rechner für jeden Schüler auch nötig.
So lange man aber mit der Planung, also der objektorientierten Analyse, dem Design und den Anforderungen beschäftigt ist, ist dies nicht so entscheidend. Hier ist durchaus Teamarbeit nützlich, so dass die Rechnerausstattung nicht so stark im Vordergrund steht. Viele

Tätigkeiten bei der Planung sind auch ohne Rechnerunterstützung möglich, so z.B. der Entwurf des Klassendiagramms, wenn auch zum Teil mühevoller. Allerdings darf man nicht vergessen, dass aus Motivationsgründen das Verwenden eines Rechners auch hier wichtig ist.

Bei den zeitlichen Ressourcen sieht es nicht so gut aus. Fast alle Lehrer kamen mit der laut Lehrplan geforderten Anzahl von Stunden nicht aus. So müsste entweder die Anzahl der Stunden im Lehrplan erhöht oder Inhalte gestrichen werden. Es könnte z.B. die 3-Schichtenarchitektur oder das Zustandsdiagramm ausgelassen werden. Die 3-Schichtenarchitektur ist ein bereits etwas tiefergehendes Konzept und das Zustandsdiagramm ist nicht unbedingt dem objektorientierten Softwareentwurf zuzuordnen, sondern ist auch in der strukturierten Programmierung ein Thema.

Zur sozialen Herkunft konnte festgestellt werden, dass sich zwischen den beiden Gruppen keine signifikanten Unterschiede ergeben. Hier wäre evtl. ein Vergleich mit der sozialen Zusammensetzung der Schüler in Gymnasien bzw. mit der Gesamtpopulation interessant.

Intelligenzquotient
Auffällig ist der recht hohe Mittelwert des Intelligenzquotienten von IQ=122 (siehe Kapitel 4.1.3) über alle Klassen, da der Mittelwert in Gymnasien 110 beträgt [We98]. Dies könnte eine Folge der Auswahl der Schüler an beruflichen Gymnasien sein. In der Regel kommen die Schüler hauptsächlich nach Abschluss der Mittleren Reife aus der Realschule, um die Studierfähigkeit zu erlangen. Sie sind damit eine Auswahl der besten Realschüler. Weiterhin werden sich bei der Schulwahl vermutlich eher technisch orientierte und begabte Schüler für das technische Gymnasium entscheiden. Diese Schüler bringen wahrscheinlich auch eher die im Intelligenztest hauptsächlich abgefragte Ebene der figuralen Ebene der Intelligenz mit. Hierzu müssten allerdings genauere Untersuchungen unter Einbeziehung allgemeinbildender Gymnasien erfolgen.

Lernerfolg und Gruppenvergleich
In der Umsetzungsgruppe wird im Vergleich zur Kontrollgruppe ein signifikant höherer Lernerfolg festgestellt (siehe Kapitel 4.3.2). Dies muss allerdings nicht das Ergebnis des Konzepts, sondern kann auch die Folge der etwas verspätet durchgeführten Vortests in der Kontrollgruppe sein (siehe Kapitel 4.1, S. 64). Deswegen darf dieses Ergebnis nicht überbewertet werden.
Für den Vergleich zwischen den beiden Gruppen ist deswegen der Nachtest wichtiger (siehe Kapitel 4.3.3). Dabei zeigt sich, egal ob univariat/multivariat oder ob unter der Annahme der Normalverteilung der Testergebnisse / ohne diese Voraussetzung untersucht wird, dass die Umsetzungsgruppe stets signifikant besser abschneidet.
Eine genauere Analyse ergibt (siehe Kapitel 4.3.4), dass diese Verbesserung vor allem in den höheren Lernzieltaxonomiestufen zu finden sind. Dies ist hinsichtlich der Zielvorgabe für ein Gymnasium, der Studierfähigkeit, recht wichtig, da die höheren Taxonomiestufen wie die Analyse, Synthese und Evaluation im Studium und auch im Berufsleben immer stärker gefordert werden.
Im Bereich der Generalisierung sind beide Gruppen gleich stark, außer bei der Evaluation, bei welcher die Umsetzungsgruppe besser abschloss. Da die Umsetzungsgruppe im Bereich

der Beziehungen über alle Taxonomiestufen hinweg signifikant besser ist, ist sie insgesamt besser befähigt, Klassendiagramme zu entwerfen und zu vergleichen, also Softwarearchitekturen zu erstellen.

Bei der durchgeführten Diskriminanzanalyse zur Trennung der beiden Gruppen korreliert das Lernziel Q stark mit der ersten Diskriminanzfunktion (siehe Tabelle 26). Dieses Lernziel bezieht sich jedoch nur auf Frage 5.4 des Fragebogens (siehe Anhang C.6), in dem Designrichtlinien abgefragt wurden. Es ist hier zu vermuten, dass dieser Inhalt in der Taxonomiestufe *Wissen* in der Kontrollgruppe nicht vermittelt wurde. Interessant ist jedoch, dass die Lernziele I und H, also die Synthese bzw. Evaluation des Inhalts *Beziehungen*, relativ weit oben in der Korrelationsrangfolge stehen. Dies bedeutet, dass die Synthese und die Evaluation von Beziehungen eine stark trennende Eigenschaft in Bezug auf die beiden Gruppen darstellt. Demgegenüber ist die Synthese einer Generalisierung (Lernziel O) nicht sehr stark trennend.
Die Diskriminanzanalyse zur Trennung der Schulklassen bestätig das oben Gesagte und hebt besonders die Mitgliedschaft zur Umsetzungsgruppe als einen wichtigen Faktor zum besseren Abschneiden des Nachtests hervor.

Zusammengefasst zeigt sich, dass der Erfolg der Umsetzungsgruppe hauptsächlich auf den höheren Lernzieltaxonomiestufen wie Synthese und Evaluation und dem besseren Abschneiden im Bereich der Beziehungen fußt. Im Bereich der Generalisierung sind sich die beiden Gruppen näher.
Insgesamt wird damit die Intention und das Ziel des Konzepts bestätigt.

Einfluss der Faktoren
Eine Analyse der Faktoren, welche das Ergebnis des Nachtests beeinflussen, kann 54% der Varianz erklären (siehe Abbildung 33). Die Zugehörigkeit zur Umsetzungs-/Kontrollgruppe ergibt dabei einen Anteil von 9% und ist der drittwichtigste Faktor. Der wichtigste Faktor ist jedoch die Fähigkeit strukturiert zu programmieren (24%). Dies könnte evtl. zeigen, dass die Schüler, welche bereits strukturierte Aufgaben besser lösen können, mit abstrakten Aufgaben besser umgehen können oder auch höher motiviert sind. Der zweitwichtigste Faktor, die subjektive Einstellung des Schülers zum Programmieren (16%), kann wahrscheinlich als Motivationsfaktor gewertet werden. Ein größerer Wunsch zum Programmieren bedeutet auch eine höhere Motivation und damit ein besseres Ergebnis im Nachtest. Allerdings spielt der Programmierwunsch vor Vermittlung der Einheit keine Rolle. Erklären lässt sich dies damit, dass der Programmierwunsch nach Vermittlung der Einheit eher die Motivation für den objektorientierten Softwareentwurf darstellt, da erst danach die Inhalte für die Schüler bekannt sind. Des Weiteren trifft die Abfrage des Wunsches nach Vermittlung der Einheit zeitlich mit dem Nachtest direkt zusammen, da dieser Wunsch direkt im Nachtest abgefragt wurde (siehe Anhang C.6).
Interessant ist, dass der Intelligenzquotient keine Rolle spielt. Dies widerspricht der Vermutung, dass abstraktes Denkvermögen für den Erfolg wichtig ist. Es könnte jedoch sein, dass Programmieren eine bestimmte, von der Intelligenz nur teilweise abhängige Fähigkeit ist bzw. nur einen Teilbereich der Intelligenz wiederspiegelt. Zu bedenken ist auch, dass die flüssige Intelligenz überprüft wurde. Hier ist aber ein methodisches Vorwissen hilfreich. Kann man mit abstrakten Themen umgehen, so ist dies evtl. bereits kristallisierte Intelligenz,

welche aber nicht ermittelt wurde und sich bereits auf konkrete Themen bezieht. Es könnte jedoch auch sein, dass dies eine Folge des recht hohen Intelligenzquotienten in allen Gruppen ist, also dass es zwar wichtig ist, eine hohe Intelligenz zu haben, dann diese Eigenschaft aber keine große Rolle mehr spielt.

Verwendet man im varianzanalytischen Modell nicht den Faktor *Umsetzungs-/Kontrollgruppe*, sondern den Faktor *Schulklasse*, ergibt sich ein anderes Bild (siehe Abbildung 34). Im Gegensatz zu oben kommen hier die Rahmenbedingungen der Schulklassen (siehe Kapitel 4.2) mehr zum Tragen. Oben wurde über jeweils 3 (Umsetzungsgruppe) bzw. 4 (Kontrollgruppe) Schulklassen gemittelt. Es zeigt sich, dass der Einfluss der Schulklasse ca. doppelt so groß ist wie der Einfluss der Mitgliedschaft in der Umsetzungs-/Kontrollgruppe. Dies ist auch verständlich, da zusätzlich zum Einfluss des Konzepts nun die Lehrerpersönlichkeit stärker zum Tragen kommt, welche bei der Einteilung in die beiden Gruppen herausgemittelt wurde.
Der Anteil der unaufgeklärten Varianz steigt aber auf 73% stark an. Dies ist vermutlich eine Folge der nun kleineren Gruppen, sodass für eine ähnlich gute gesamte Varianzaufklärung wie oben mehr Teilnehmer nötig gewesen wären. Unterstützt wird diese Vermutung dadurch, dass der Faktor *Ergebnis der Struktogrammaufgabe*, welcher oben noch signifikant war, hier ein α-Fehler von 7,9% besitzt, bei einer Varianzaufklärung von 4,4%.

Für beide Analysen ist es wichtig, dass die Varianzen homogen sind. Ein Levene-Test ergab für die Untersuchung unter Verwendung des Faktors *Umsetzungs-/Kontrollgruppe* eine Signifikanz von 13,6% unter Verwendung des Faktors *Schulklasse* eine Signifikanz von 0,8%[49]. Dies bedeutet, dass bei der Ermittlung der Varianzaufklärung die Voraussetzungen verletzt sind. Vermutlich sind die Stichproben, vor allem bei Einteilung in die 7 Schulklassen, zu klein.

Um der Problematik der impliziten Gewichtung der Lernziele zu entgehen wurde auch für die Untersuchung der Einflussfaktoren multivariat analysiert (siehe Tabelle 27). Um des Weiteren unabhängig von der Normalverteilung der Lernziele zu sein wurde zusätzlich das verteilungsfreie Verfahren nach Zwick [Zw85] verwendet[50]. Wie bei der univariaten Analyse ergab sich dabei das selbe Problem, dass die Faktoren *Umsetzungs-/Kontrollgruppe* und *Schulklasse* nicht gemeinsam verwendet werden konnten. So wurden insgesamt vier multivariate Untersuchungen durchgeführt:

- unter Verwendung des Faktors *Umsetzungs-/Kontrollgrupp*e bzw. *Schulklasse* und
- unter der Voraussetzung der Normalverteilung der einzelnen Lernziele bzw. ohne diese Voraussetzung.

Es zeigt sich, dass sowohl die Schulklasse als auch die Mitgliedschaft in der Umsetzungsgruppe einen signifikanten Einfluss besitzen. Bei allen vier multivariaten Untersuchungen ist auch das Programmieren in der Freizeit signifikant, was einerseits in Bezug auf die obige univariaten Varianzanalyse überrascht, da dort dieser Faktor keine Rolle spielte. Auf der anderen Seite lässt sich dies gut erklären, da Übung und Erfahrungen beim Programmieren

49 Beim Levene-Test ist eine homogene Verteilung bei einer Signifikanz größer 20% gegeben [Po02, S6-50].

50 In diesem Fall spielt auch die Homogenität der Varianzen keine Rolle mehr.

zu Hause die Fähigkeiten und Kenntnisse verbessern. In diesem Zusammenhang ist wahrscheinlich auch der Faktor *Programmiersprache zu Hause vorhanden* zu sehen.
Der Faktor *Programmierwunsch nach Vermittlung der Einheit* ist nun nicht mehr signifikant, dafür der Faktor *Ergebnis der Struktogrammaufgabe*. Dies könnte evtl. auf eine spezielle Fähigkeit zum abstrakten Denken im Bereich der Informatik hindeuten, wie er oben bereits erwähnt wurde. Es könnte jedoch auch sein, dass dies aus einer Korrelation zum Programmieren in der Freizeit herrührt, da bei dieser Freizeitbeschäftigung sicher auch das Denken in Algorithmen geschult wird. Allerdings ist diese Signifikanz nur bei einer Unterteilung in die Umsetzungs-/Kontrollgruppe signifikant, nicht bei einer Unterteilung in Schulklassen.
Bestätigt wird jedoch wiederum der fehlende Einfluss des Intelligenzquotienten, der Freizeitbeschäftigung mit dem Rechner und des Spielens in der Freizeit mit dem Rechner. Überraschenderweise scheint die flüssige Intelligenz, wie sie hier getestet wurde, keine Rolle zu spielen, allerdings ist der Intelligenzquotient recht hoch. Die anderen beiden Faktoren sagen klar, dass irgendeine Beschäftigung mit dem Rechner oder gar das Spielen mit dem Rechner keinen Einfluss auf das Ergebnis hatten, also in Bezug auf den objektorientierten Softwareentwurf bedeutungslos sind.
Das multivariate und verteilungsfreie Verfahren benötigt in Bezug auf die Voraussetzungen die wenigsten Anforderungen. Sie liefert zwar nicht so viele Informationen, wie z.B. die univariate Varianzanalyse mit ihrer Varianzaufklärung, aber die Ergebnisse sind am verlässlichsten.
So ist abschließend zu sagen, dass das Ergebnis des Nachtests einerseits von der Umsetzungs-/Kontrollgruppe bzw. der Schulklasse abhängt, des Weiteren davon, dass zu Hause die Möglichkeit besteht, zu programmieren und dies auch wirklich gemacht wird, nicht jedoch von der flüssigen Intelligenz oder dem Wunsch zu programmieren.

Einschätzung der Lehrer
Der Unterricht scheint das Selbstbewusstsein der Lehrer im fachlichen und pädagogischen Bereich des objektorientierten Softwareentwurfs verbessert zu haben, unabhängig davon, ob am Konzept teilgenommen wurde oder nicht. Interessanterweise ist der absolute Betrag der Verbesserung in beiden Gruppen identisch, sowohl im fachlichen als auch im pädagogischen Bereich (siehe Kapitel 4.3.6.1).

Der Unterricht scheint allen Lehrern Spaß zu machen, wobei in der Umsetzungsgruppe dieser etwas größer war. Die Motivation der Schüler aus der subjektiven Sicht der Lehrer scheint nahezu identisch zu sein. Allerdings scheinen die Schüler aus Sicht der Lehrer in der Umsetzungsgruppe wesentlich besser für das Abitur und ein Studium gerüstet. Vor allem die Vorbereitung auf das Abitur scheint in der Umsetzungsgruppe eklatant besser zu sein.

Einschätzung der Schüler
Die Schüler der Umsetzungsgruppe sehen im Vergleich zur Kontrollgruppe genau dort ihre Stärken (siehe Kapitel 4.3.6.2), in denen sie auch im Nachtest besser abgeschnitten haben: Beziehungen und Klassenentwurf. In Bereichen, in denen sie dagegen in etwa dieselben Ergebnisse hatten, sehen die Schüler der Umsetzungsgruppe eher ihre Schwächen: Generalisierung, UML.

Dies lässt sich mit einer Verschiebung des Wertmaßstabs der Schüler erklären. Es sollten jeweils die beiden besten und die beiden schlechtesten Bereiche angegeben werden, d.h. die beiden Enden des subjektiven Maßstabs. Da die Schüler einige Bereiche subjektiv besser einschätzen, müssen in den Anderen zwangsläufig relativ zu den besseren Bereichen die Schlechten liegen.
Nimmt man die Ergebnisse des Nachtests hinzu, so folgt daraus, dass die Selbsteinschätzungen der Schüler sowohl in der Umsetzungs- als auch in der Kontrollgruppe recht gut sind. Aus dem Nachtest ist bekannt, dass die Umsetzungsgruppe in den Bereichen Beziehungen, Klassenentwurf und Use Cases besser ist.
Die Schüler bestätigen also subjektiv, dass die Stärke des hier vorgestellten Konzepts speziell in den Bereichen Beziehungen und Klassenentwurf liegt.

Der Wunsch zu programmieren wurde vor und nach der Einheit erfragt. Zwischen den beiden Gruppen besteht sowohl vor als auch nach Vermittlung der Einheit kein signifikanter Unterschied.
In beiden Gruppen verschlechterte sich jedoch dieser Wunsch zwischen Beginn und Ende der Unterrichtseinheit. Allerdings konnte nur in der Kontrollgruppe eine signifikante Verschlechterung nachgewiesen werden. In der Umsetzungsgruppe blieb diese Veränderung ohne Signifikanz. Woher diese Verschlechterung herrührt, ist nicht ganz klar. Evtl. steht der strukturierte und planvolle Ansatz, der beim objektorientierten Softwareentwurf gefordert wird, dem mehr Spaß-betonten *drauf-los*-Programmieren entgegen. Aber genau dieser strukturierte Ansatz ist ein wichtiges Lernziel.

Informatische Vorerfahrung
Die informatische Vorerfahrung zeigt im Vergleich der beiden Gruppen keinen signifikanten Unterschied (siehe Kapitel 4.3.8). Allerdings zeigt sich ein signifikanter Unterschied bei der Abhängigkeit von der Schulklasse, was bedenklich ist. Für einen Schüler bedeutet dies, dass für seinen Lernfortschritt primär die Auswahl der Schulklasse entscheidend ist.
Der starke Unterschied zwischen den Schulklassen F1 und F2 an derselben Schule ist vermutlich die Folge unterschiedlicher Lehrer in Schulklasse 11.
Es zeigt sich, dass auch im Bereich der Algorithmik, welcher ebenfalls geeignet erscheint, problemlösendes Denken zu schulen, Unterrichtskonzepte entworfen werden müssen. Das schwache Abschneiden der Schüler nach einem Jahr Unterricht (in Klasse 11: 30 Schulstunden im Fach *Informationstechnik* [It01], 60 Schulstunden im Fach *Angewandte Informationstechnik* [Ai01]) zeigt dies.

5.2 Resümee
Für die Vermittlung des pädagogischen und fachlichen Know-Hows zeigte sich, dass das in dieser Arbeit erstellte Konzept nützlich ist. Es bietet einerseits fachliche, aber vor allem auch methodische und didaktische Hinweise zur Vermittlung des objektorientierten Softwareentwurfs.
So scheint das Vorgehen entgegen dem realen Softwareentwurf, also entgegen der Reihenfolge: Anforderung, Analyse, Design, Implementierung bei der Vermittlung des objektorientierten Softwareentwurfs gut geeignet zu sein. Auch der ständige Bezug zwischen dem Modell in Form von UML-Diagrammen und Programmcode scheint geeignet zu sein und hilft, am Ende der Unterrichtseinheit, auch bei der Ermittlung der Anforderungen.

In der Evaluation bestätigte sich dies durch ein signifikantes ca. 10%ig besseres Abschneiden der Schüler im Nachtest, dies bei ähnlichen Randbedingungen der Zusammensetzung der sozialen Schichten, des Intelligenzquotienten oder den Vorerfahrungen der Lehrer und der informatischen Vorerfahrung der Schüler. In der Umsetzungsgruppe sind zwar mehr heimische Rechner vorhanden, allerdings beschäftigen sich in der Freizeit mehr Schüler aus der Kontrollgruppe mit dem Programmieren, so dass diese Randbedingungen sich in etwa ausgleichen.
Die Diskriminanzanalyse bestätigte dabei, dass das Konzept ein wichtiger Beitrag für dieses bessere Abschneiden darstellt. Allerdings ist natürlich auch der unterrichtende Lehrer eine wichtige Größe für den Erfolg.
Es zeigt sich weiterhin, dass das hier vorgestellte Konzept vor allem in den höheren Lernzieltaxonomiestufen, wie Synthese und Evaluation, seine Wirkung entfaltet und also besonders die Problemlösefähigkeit im Bereich des Softwareentwurfs fördert. Damit ist es hervorragend geeignet die Kompetenzen welche in der Pisa-Studie eingegefordert werden zu fördern.

Interessanterweise spielt für die Ergebnisse der Evaluation der Intelligenzquotient kaum eine Rolle. Allerdings ist das Intelligenzniveau in allen Schulklassen recht hoch. Es scheint so, dass eine höhere Motivation der Schüler wichtiger ist. Durch den recht hohen Intelligenzquotienten in den untersuchten Schulklassen gilt diese Aussage zumindest für Schüler mit einer höheren Intelligenz, bei schwächer begabten Schülern müsste diese Aussage noch genauer untersucht werden.
Die Unabhängigkeit vom Intelligenzquotienten ist Hoffnung und Unglück zugleich. Es bedeutet, dass durch motivierenden Unterricht noch einiges zu erreichen wäre, bedeutet aber auch, dass die Lehrer mehr Unterstützung im fachlichen und fachdidaktischen Bereich benötigen. Erst pädagogische und fachliche Hilfe von Außen bringt sie in die Lage, dies auch besser umzusetzen.

Da anscheinend die Motivation der Schüler wichtig ist, sollte überlegt werden, wie diese verbessert werden kann. Eine höhere Motivation ist bei den Schülern immer dann festzustellen, wenn am Ende ein attraktives Produkt steht. In der Softwaretechnik bedeutet dies, dass nicht nur Softwaremodelle in Form der UML erzeugt, sondern diese auch als Programm umgesetzt werden. Deswegen sollte das Codieren ein integraler Bestandteil des Unterrichts sein, selbst dann, wenn es nicht direkt als Lernziel angestrebt wird. Weiterhin können die Schüler damit ihre Modelle leichter evaluieren und testen.
Die Verbindung der Modellierung zusammen mit deren Umsetzung in Programmcode ist deswegen sicher eine der Stärken des Konzepts. Dadurch werden die zum Teil abstrakten Inhalte immer wieder real begreifbar und die Motivation verbessert.

Als Konsequenz aus der Forderung zu mehr Unterstützung der Lehrer im Bereich des Softwareentwurfs wurde das Konzept bereits in einer mehrtätigen Lehrerfortbildung vermittelt und gut angenommen. In der Zwischenzeit haben einige Lehrer begonnen, das Konzept in ihrem Unterricht umzusetzen und weiterzuentwickeln. So sind weitere Fortbildungen entstanden.

Es ist auch noch eine vielfältige Weiterentwicklung nötig: z.B. weitere Beispiele und Aufgaben, evtl. eine Integration strukturierter Programmierparadigmen wie der Vermittlung von Abfragen oder Schleifen in das Unterrichtskonzept[51] usw.
Wie Kapitel 4.3.8 zeigt sind auch Anstrengungen im strukturierten Programmieren nötig. Auch dort werden hohe Lernzieltaxonomien gefordert, wenn z.B. ein Algorithmus gesucht und dann mit anderen Lösungen verglichen werden soll.

Diese Arbeit war ein erster erfolgreicher Schritt, den objektorientierten Softwareentwurf und das damit verbundene Softwareengineering im schulischen Umfeld zu vermitteln. Es bietet einen hervorragenden Weg die z.B. in der Pisa-Studie verlangte Modellbildungskompetenz und Problemlösefähigkeit der Schüler, hier im Bereich der Informatik, zu verbessern. Weitere Schritte sind jedoch nötig.

51 Im hier bestehenden Konzept wurde dies als Eingangsvoraussetzung festgelegt.

A Unterrichtseinheiten

A.1 Einheit: Objekt und Klasse

A.1.1 Motivation

Programm *Willis Welt* (WillisWelt v1.0) vorstellen (Software siehe Anhang D). Willi und Siggi können sich nicht bewegen, nur zeigen und verstecken. Beide sind jedoch einerseits neugierig und möchten sich zum Ort des Anklickens bewegen, andererseits möchten sie sich auch näher kommen. Wir sollen das programmieren!

A.1.2 Ablauf: Programm analysieren

- gestartetes Programm *Willis Welt* (WillisWelt v1.0) betrachten (siehe Abbildung 16)
- Welche Objekte gibt es im Programm? → Willi, Siggi, Welt (siehe Abbildung 40)
- Welche Eigenschaften haben diese Objekte? → siehe Abbildung 40
- Willi und Siggi sind zwar unterschiedliche Objekte. Was ist Ihnen aber gemeinsam? → gleicher Bauplan ⇒ Klassendiagramm → Abbildung 41

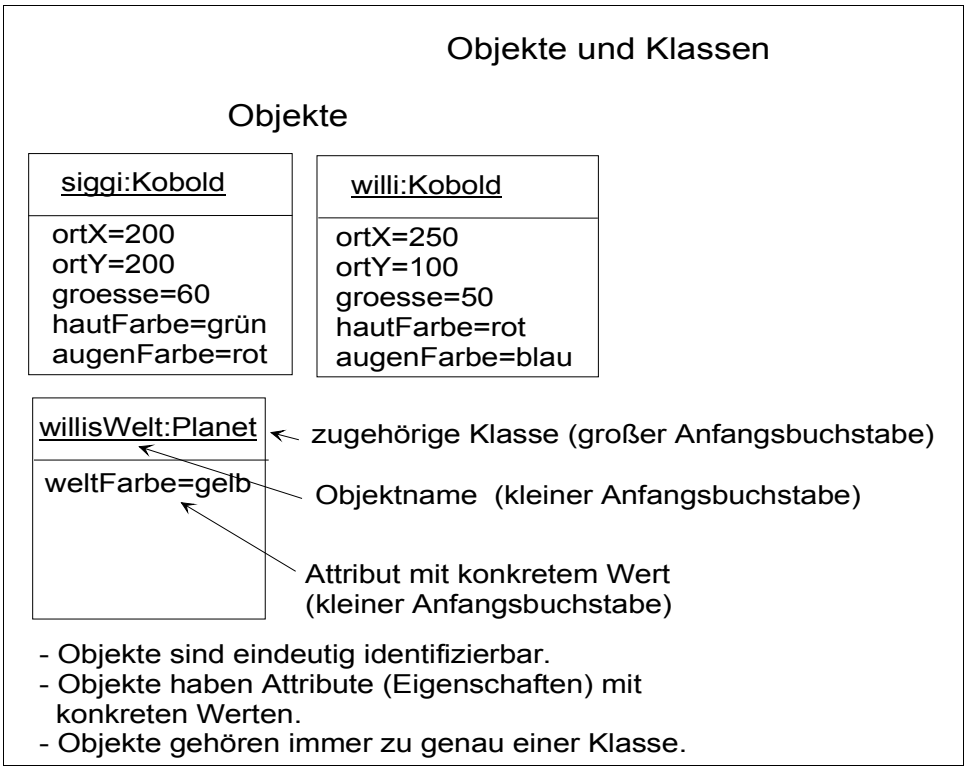


Abbildung 40. Zusammenfassung zum Objektdiagramm

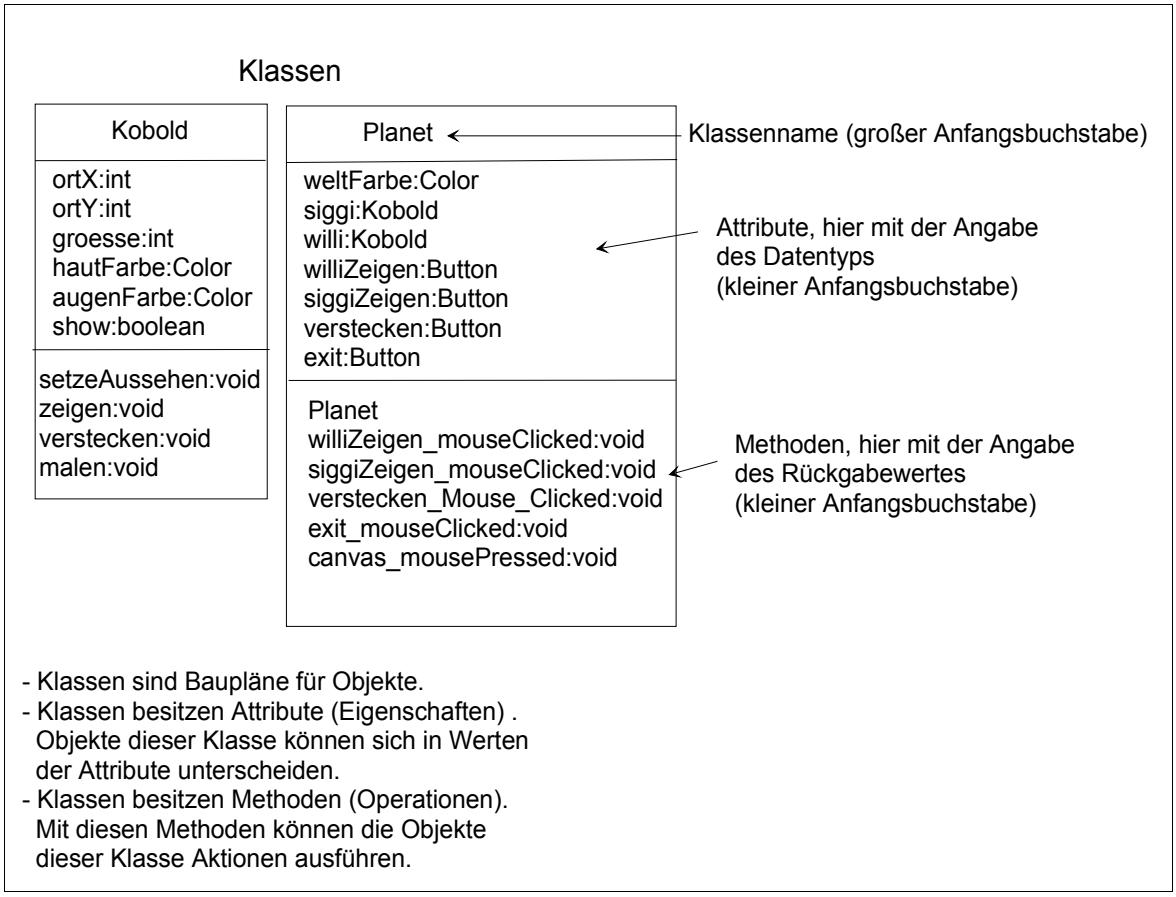


Abbildung 41. Zusammenfassung zum Klassendiagramm

- Suchen weiterer Beispiele für Objekte und Klassen durch die Schüler (Beispiele siehe Abbildung 42, zuerst nur eine Sammlung in Textform!)

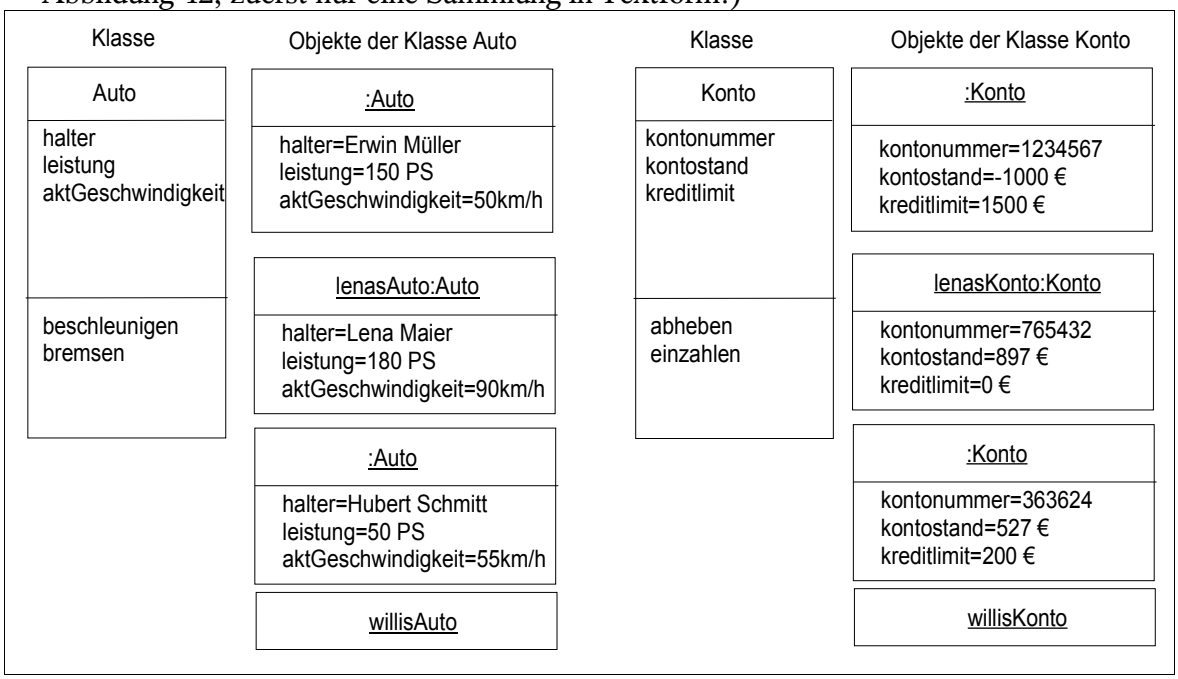


Abbildung 42. Weitere Beispiele für Objekte und Klassen

- Welche Gemeinsamkeiten gibt es im Programm *Willis Welt* neben den gleichen Attributen noch? → Verstecken/Zeigen; Objekte können auch Aktivitäten ausführen ⇒ Methode/Operation im Klassendiagramm (siehe Abbildung 41)
- Suchen von Methoden und Attributen zu den oben gefundenen Klassen (siehe Abbildung 42)

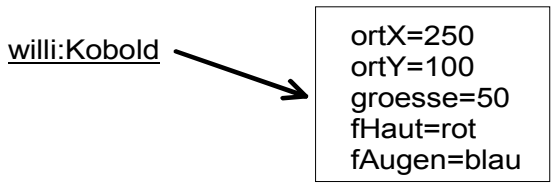
- Den Code des Programms austeilen
  Die Schüler sollen die Attribute und die Methoden aus dem Klassendiagramm im Code farbig markieren (Hinweis: die Einheit class ... {...} ist eine Klasse, dies ist der Bauplan für einen Kobold). Kurzes Eingehen auf die Funktionalität einzelner Methoden (*zeigen()/verstecken()*, Aufgabe der Klassenmethoden *repaint()* und *paint()*)

Referenzen und Objekte in Java

Durch die Zeile
`Kobold willi;`
wird nur eine Referenz auf ein Objekt vom Typ `Kobold` angelegt.
Das Objekt exisiert aber noch nicht!

In der Zeile
`willi = new Kobold();`
wird ein neues Objekt erzeugt. Die Referenz `willi` zeigt auf dieses Objekt.

Beides kann man auch in einer Zeile verbinden:
`Kobold willi = new Kobold();`



Beispiel:
```
1    Kobold willi, hugo;
2    willi = new Kobold();
3    hugo = willi;
```

Es gibt nur ein Objekt vom Typ `Kobold`. Beide Referenzen (`willi` und `hugo`) zeigen auf das selbe Objekt!

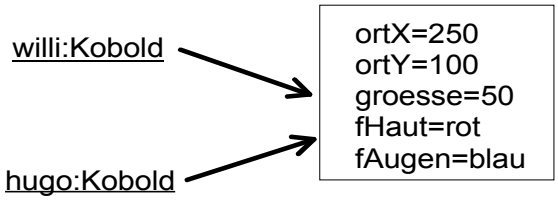


Abbildung 43. Objekte und Referenzen auf Objekte

- Wo werden Objekte (hier: Kobolde) erzeugt? → Zeile *willi = new Kobold()* (siehe Abbildung 43)
- Welche Bedeutung hat die Zeile *Kobold willi*? → Erklärung des Unterschieds zwischen Referenz auf ein Objekt und dem Objekt (siehe Abbildung 43)

- In einer Vertiefungsstunde bzw. in CT kann man auf den Unterschied *Call by Value* und *Call by Reference* eingehen. Dieser Unterschied tritt auch in der strukturierten Programmierung auf und ist sprachabhängig, weswegen hier nicht weiter darauf eingegangen wird. In C++ müsste weiterhin auf das Problem der Zeiger eingegangen werden.

A.1.3 Ablauf: Programm anpassen

- gestartetes Programm, in dem Willi und Siggi gehen können (WillisWelt v1.1) betrachten
- Was muss man der Klasse Kobold hinzufügen, damit die Kobolde gehen können? → Attribut s*chrittweite* und Methode *gehen(...)*. Einzeichnen in das Klassendiagramm (siehe Abbildung 44). Dabei sollte man auch auf die Parameterübergabe eingehen (Wohin soll der Kobold gehen?). Entwicklung der Methode *gehen()* durch die Schüler in Kleingruppen. Einige Dinge müssen aus den Methoden *williZeigen_mouseClicked()* / *verstecken_mouseClicked()* abgeschaut werden. Vorstellen im Sequenzdiagramm (siehe Abbildung 45); Details hierzu werden erst später vermittelt. Der Aufruf des Betriebssystems ist hier eine didaktische Reduktion. Das Betriebssystem meldet die Ereignisse an das Programm, welches diese dann alle selbst verarbeitet. Diese Funktionalität ist von einer Oberklasse geerbt.
- evtl. sollte hier das Thema *Ereignissteuerung* vertieft werden
- codieren und testen der Veränderungen im Programm durch die Schüler (ausgehend von v 1.0)

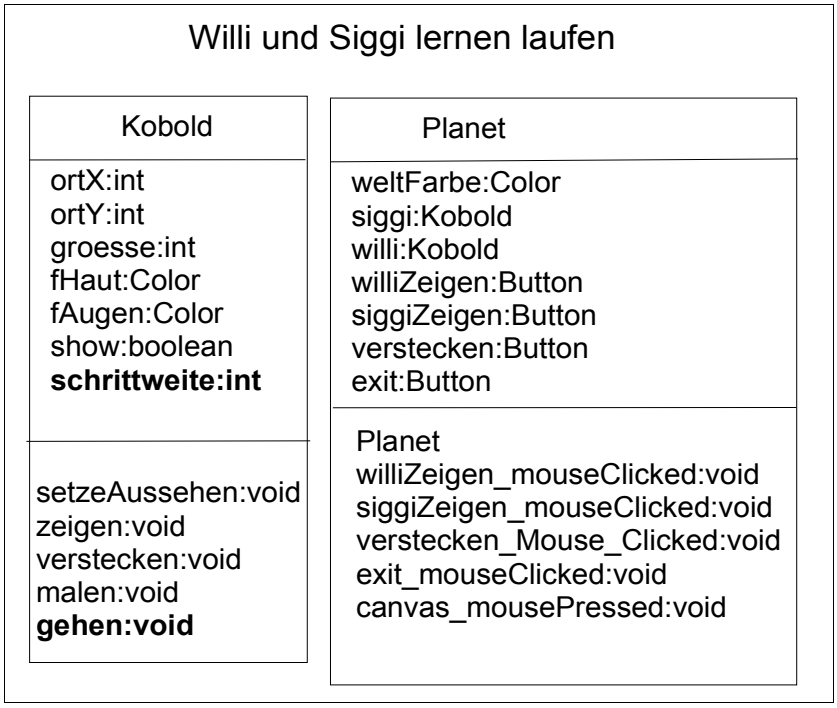


Abbildung 44. Die neuen Attribute und Methoden in den Klassen

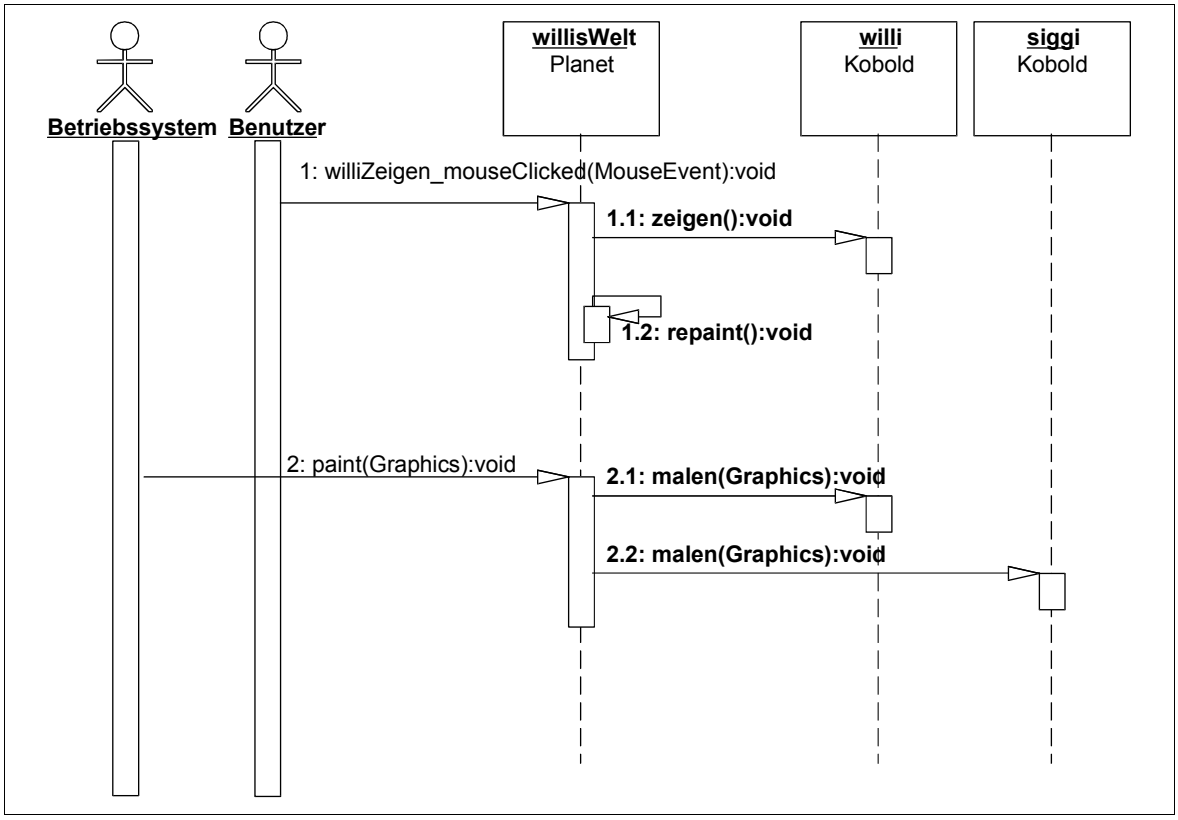


Abbildung 45. Der Ablauf der Ereignisse im Sequenzdiagramm

- Verbesserung von gehen: **neben** der Methode *gehen(int x, int y)* soll nun auch die Methode *gehen(float phi)* (gehe phi Grad mit der Größe *schrittweite*; 0° → Norden) existieren. Geht das? → Zeigen im Klassendiagramm (siehe Abbildung 46)
  Wie kann der Compiler beide unterscheiden? → aufgrund der Parameter und deren Reihenfolge. Der Algorithmus soll von den Schülern entworfen werden, evtl. Hilfestellung durch eine Skizze (siehe Abbildung 46)
  evtl. mathematische Hilfestellungen anbieten
- Entwurf, programmieren und testen des Algorithmus in Schülergruppen (siehe Abbildung 46) (Lösung: WillisWelt v1.3)

Ablauf: Konstruktor und Zugriffsrechte

- bei der Geburt eines Objekts sollen Attribute festgelegt werden, z.B. die Hautfarbe eines neuen Kobolds. Wie kann man bei der Geburt eines Objekts Attribute festlegen? → Konstruktor, eine Methode, welche bei der Geburt eines Objekts aufgerufen wird. Sie hat denselben Namen wie die Klasse.
  Hier auch evtl. Destruktor zum Aufräumen interner Objekte. Dieser wird beim Sterben eines Objekts aufgerufen (in Java nicht implementiert, die Methode finalize() der Klasse *Object* entspricht grob diesem Verhalten. Sie wird aber erst bei der Garbage-Collection aufgerufen!) (siehe Abbildung 47).
- Soll man die Hautfarbe und die Augenfarbe nach der Geburt ändern können? Nein → Information Hiding, Kapselung. Auch der Aufenthaltsort darf nicht wild verändert werden, sondern nur schrittweise! (siehe Abbildung 47)
  Die Verwendung der Zugriffsrechte im Programmcode sollte direkt im Programm gezeigt werden.

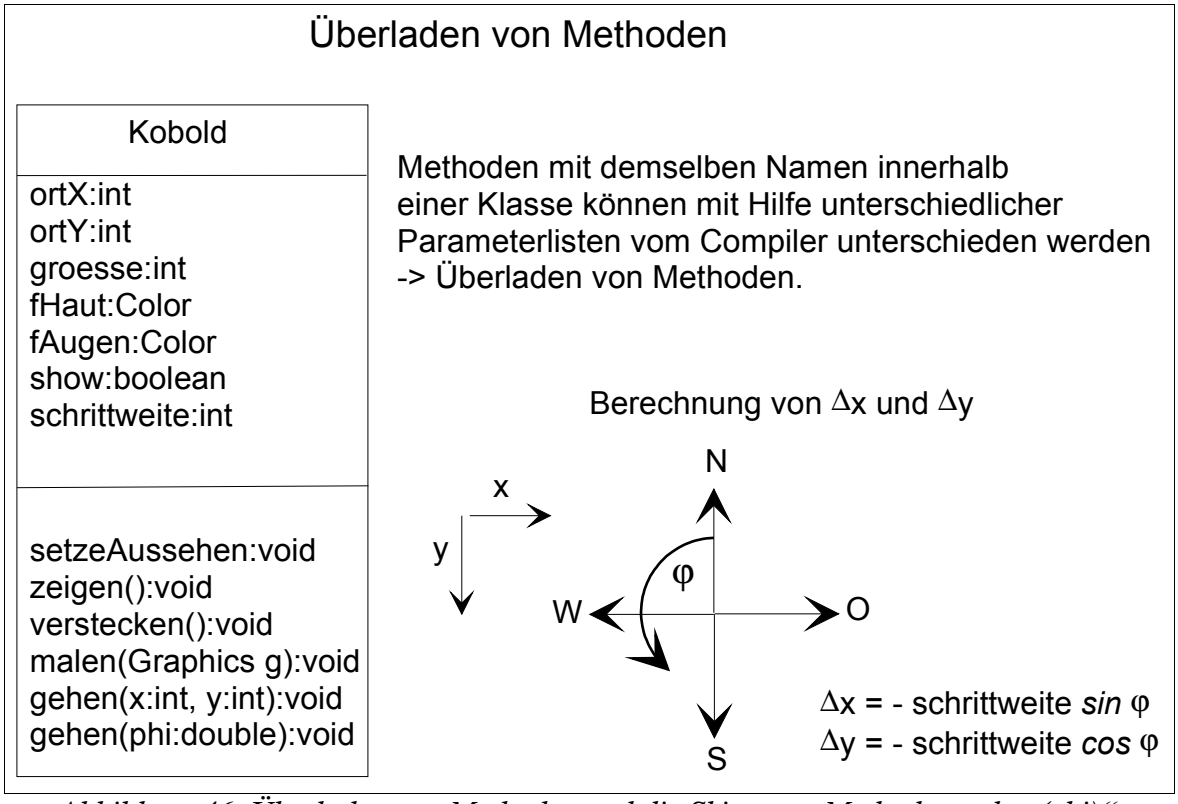


Abbildung 46. Überladen von Methoden und die Skizze zur Methode „gehen(phi)“

- Weiteres Beispiel:
  Temperaturklasse: Ob eine Klasse die Temperatur in Celsius oder Fahrenheit speichert, ist von außerhalb der Klasse nicht relevant. Die konkrete Art der Speicherung kann geändert werden, ohne die Zugriffsmethoden zu ändern.
- Anpassung des Programms durch die Schüler (Lösung *WillisWelt v1.5*)
  Die Lösungen sollten eingesammelt werden. Bei der Korrektur auf guten Programmierstil achten (keine/zu viele/schlechte Kommentare, eigentlich lokale Variablen werden zu Attributen, ...). Eine gute Dokumentation sollte so geschrieben sein, dass ein fremder Informatiker den Aufbau des Programms möglichst rasch versteht!
  Korrektur mit Kommentaren zurückgeben

- Programmieren einer Klasse (Klassendiagramm gegeben, graphische Methoden bereits programmiert) und erzeugen von Objekten (evtl. in CT)
  Z.B. Autos fahren lassen etc. Darauf achten, dass Attribute gekapselt sind und auf eine gute Dokumentation (Klassendiagramm, Kommentare im Quellcode, textuelle Beschreibung der Aufgaben von Attributen und Methoden) Wert legen

Konstruktor

Der Konstruktor ist eine Methode, welche bei der Geburt
eines Objekts aufgerufen wird.
Einem Konstruktor können Parameter mitgegeben werden.
Er hat denselben Namen wie die Klasse.
Beispiel in Java:

Deklaration:

```
class Temperatur {
   private  double  celcius;
  Temperatur(double anfangsTempInC)  {
         celcius=anfangsTempInC;
   }
}
```

Verwendung:

```
Temperatur eineTemperatur = new Temperatur(10.6);
```

Ein Destruktor wird beim Sterben eines Objekts aufgerufen.
Mit ihm kann man vorher noch Dinge in Ordnung bringen,
z.B. eine Hilfsdatei löschen.
(In Java ist kein Destruktor implementiert!)

Datenkapselung, Information Hiding

Daten sollen vor dem direkten Zugriff von außen, d.h. außerhalb der eigenen Klasse, geschützt sein. So kann der Zugriff auf die Daten besser gesteuert werden.
Beispiele:
- Eine Variable soll nur lesbar, aber nicht zu verändern sein (z.B. die Hautfarbe des Kobolds).
- Die interne Datenorganisation kann geändert werden, ohne dass andere Klassen geändert werden müssen. Nur die Deklaration der Zugriffsmethoden muss gleich bleiben! (z.B. Klasse zur Verwaltung von Temperaturen in Celcius und Fahrenheit).

Zugriffsrechte (Sichtbarkeit, visibility):
- \- private Nur klassenintern zugreifbar
- \+ public klassenintern und -extern zugreifbar

In der Regel sind Attribute *private* und Methoden *public* deklariert.

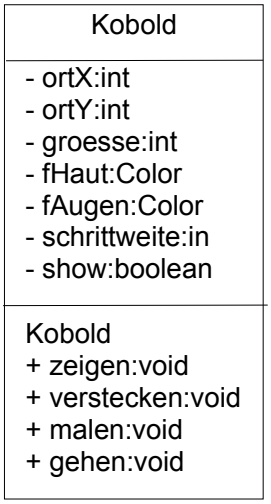


Abbildung 47. Der Konstruktor, Destruktor und die Datenkapselung

A.2 Einheit: Beziehungen

A.2.1 Motivation

Das bisher programmierte Programm (WillisWelt v1.5) vorführen. Die beiden Kobolde fühlen sich einsam. Wir wollen nun ein kleines Volk von Kobolden zum Leben erwecken und dies programmieren.

A.2.2 Ablauf: Assoziationen und Kardinalitäten

- Die beiden Klassen *Kobold* und *Planet* zeigen (ohne Attribute und Methoden; siehe Abbildung 48). Wir wissen, dass eine Klasse aus Methoden und Attributen besteht.
- Wie stehen Klassen untereinander in Verbindung?
- Welche Verbindung besteht zwischen der Klasse *Kobold* und der Klasse *Planet*?
  → Klassen können untereinander Beziehungen haben. Um diese Beziehung besser zu erkennen, kann die Beziehung (Assoziation) mit einem Verb bezeichnet werden (siehe Abbildung 48).
- Jeder Kobold soll genau auf einem Planet leben. Auf jedem Planeten sollen maximal 4 Kobolde leben (Schutz vor Überbevölkerung). Wie notiert man das im Klassendiagramm? → Kardinalität, siehe Abbildung 48

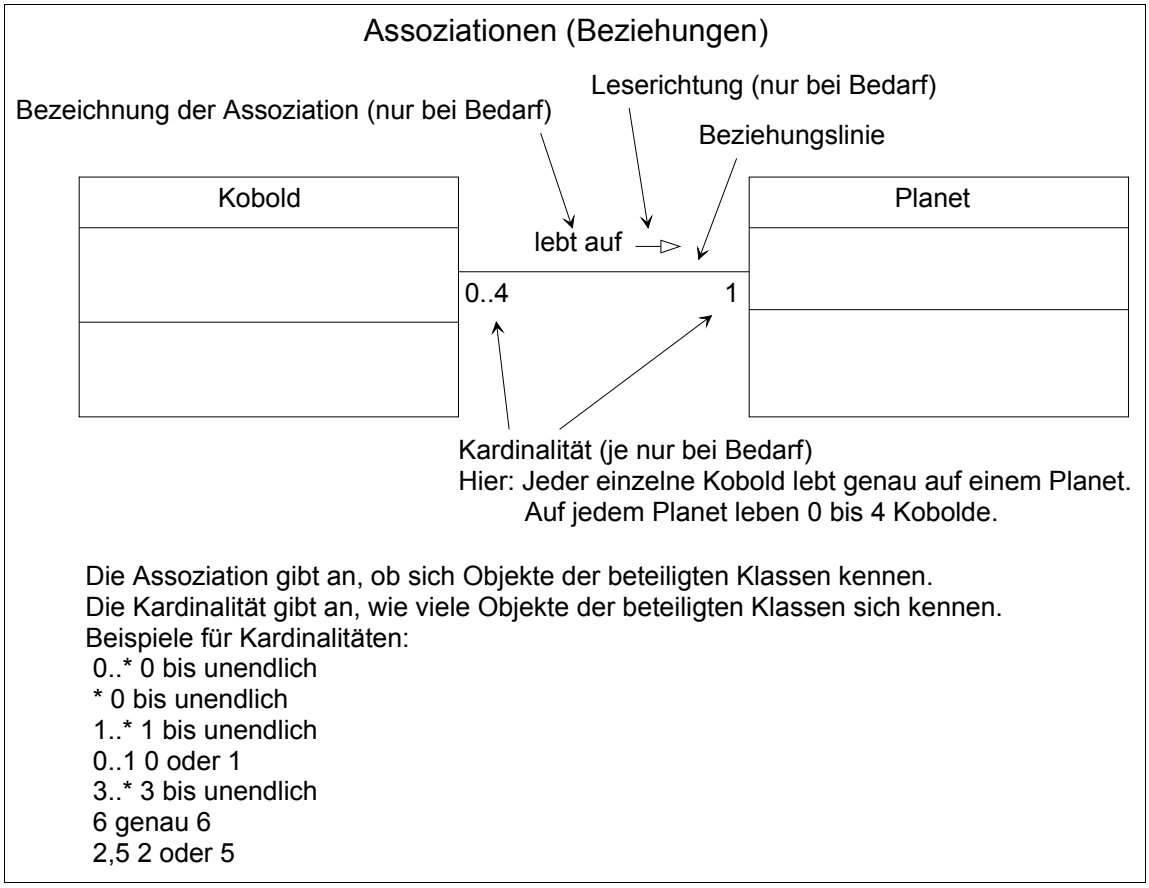


Abbildung 48. Beziehungen zwischen Klassen

- Arbeitsblatt 1 (siehe Anhang B.1):
  1. Einfache Beispiele für Assoziationen und Kardinalitäten. Bei der Besprechung ist eine Diskussion der Lösungen sehr wichtig. Hier gibt es nicht **die** Lösung, z.B. kann man für die Kardinalität der Reifen (Aufgabe a) 4 oder 3,4 oder 3..5 angeben,

je nachdem welche Sonderfälle berücksichtigt werden (dreirädrige Autos, gehört das Ersatzrad hinzu, ...).

2. Wie wird eine Assoziation im Code realisiert? Die Assoziation wird über ein Attribut in einer (oder beiden) Klassen realisiert. In welcher Klasse dies geschieht, ist meist erst für die Implementierung wichtig. Um die Klasse zu kennzeichnen, kann man einen Pfeil an die Assoziationslinie anbringen, welche die Navigierbarkeit symbolisiert (siehe Abbildung 49).
3. Wie kann man eine Kardinalität 0..4 im Programmcode realisieren? → Array of Kobold mit 4 Zellen ; Programmieren des Beispiels auf 4 Kobolde (unendliche bzw. unbekannte Kardinalität später); Fertigstellung evtl. in CT; mögliche Lösung siehe die Softwareversion WillisWelt v1.7 der Kobolde.

Nach jeder Aufgabe Besprechung dieser.

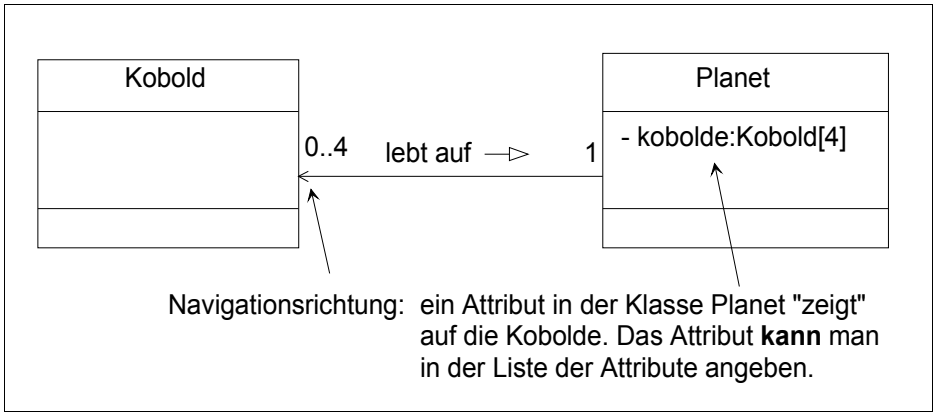


Abbildung 49. Navigierbarkeit im Klassendiagramm

- Arbeitsblatt 2 (siehe Anhang B.2):
  Schüler sollen Beziehungsnamen und Kardinalitäten finden. Die Klassen sind gegeben, Assoziationen sollten einfach bzw. auch bereits gegeben sein. Hierbei handelt es sich um die Analyse eines Problems aus der realen Welt. Die Frage *WAS* steht im Vordergrund. Der Begriff objektorientierte Analyse (OOA) wird eingeführt (siehe Abbildung 50).

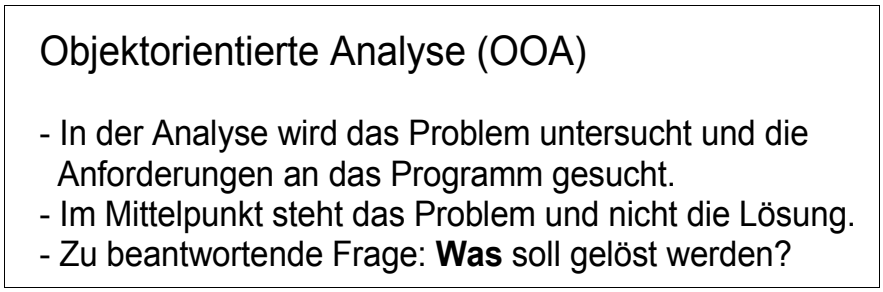


Abbildung 50. Der Begriff der objektorientierten Analyse

- Arbeitsblatt 3 (siehe Anhang B.3; entnommen aus [Br01]) als weitere, tiefer gehende Übung (evtl. als Hausaufgabe).

A.2.3 Ablauf: Rollen

- Die Kobolde wollen einen König. Er ist ein normaler Kobold und hat eine Krone auf dem Haupt. Auf jedem Planet gibt es genau einen König mit **einer** Krone. War das Volk mit dem König nicht zufrieden, kann ein neuer König ernannt werden.
  Wie müssen wir das Klassendiagramm erweitern?
  → Neue Klasse *Krone*, diese hat eine Beziehung zur Klasse *Planet* und Klasse *Kobold*.
  Einführung des Rollennamens (siehe Abbildung 51)

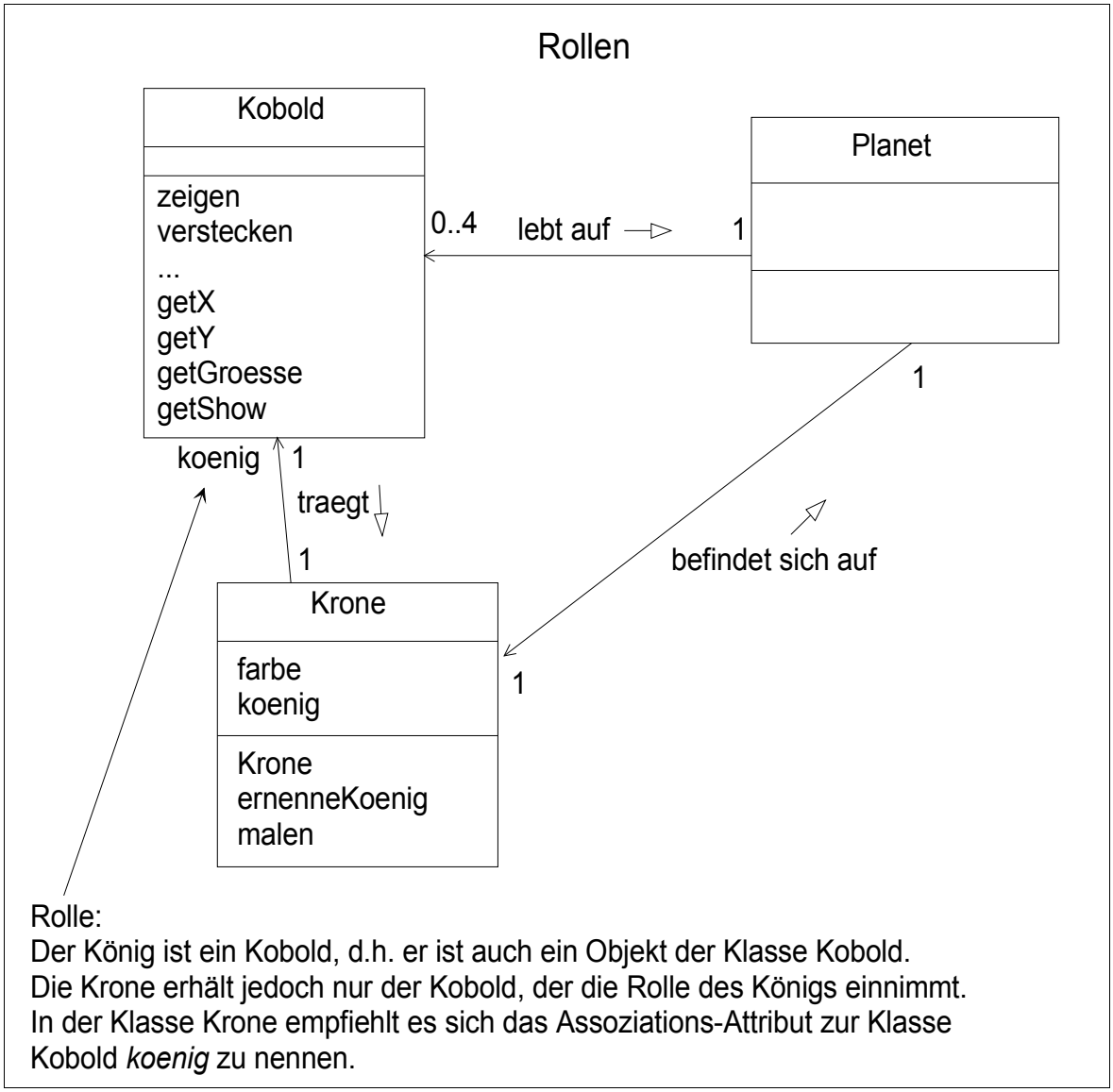


Abbildung 51. Rollen im Klassendiagramm

- Suchen weiterer Beispiele mit Rollen (siehe Abbildung 52)
  - Welche *Rollen* spielen sie im Leben? → Schüler, Sohn/Tochter, Freund/Freundin, ...
  - Verwaltung eines Firmenparkplatzes: Es gibt Firmenfahrzeuge und Privatfahrzeuge.
- Welche Eigenschaften und Methoden benötigt die neue Klasse *Krone*? Wie müssen wir evtl. die Klasse *Kobold* und *Planet* erweitern?

→ Hier wird ein wichtiger Schritt beim Klassendesign beschritten. Welche Aufgaben und Verantwortlichkeiten hat die Klasse? Welche Attribute und Methoden benötigt sie dafür?
→ siehe Abbildung 51; die Änderungen können auch direkt mit einem UML-Tool (z.B. Together [To02]) eingetragen werden. Wenn nicht bereits erfolgt, kann hier eine kurze Einführung in das verwendete UML-Tool erfolgen, sodass die Schüler die Änderungen über das Tool in ihr Programm einfügen und danach die Methoden formulieren und das Programm schreiben können.

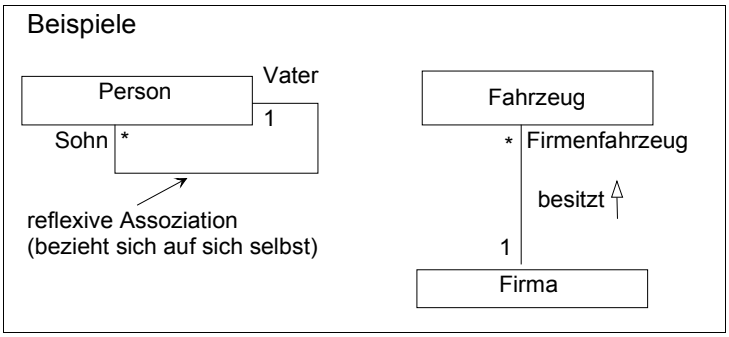


Abbildung 52. Weitere Beispiele für Rollen

- Programmierung des Problems (die Methode *malen()* der Klasse *Krone* sollte vorgegeben werden; Lösung WillisWelt v1.9, siehe Abbildung 19)

A.2.4 Ablauf: Aggregation und Komposition

- Wenn der Planet des Kobolds zerstört wird, stirbt auch der Kobold. Wie zeigt sich das im Klassendiagramm?
  → Komposition, Aggregation (siehe Abbildung 53)
- Weitere Beispiele zur Aggregation und Komposition sollen von den Schülern bearbeitet werden (mögliche Lösung siehe Abbildung 54):
  - Datei – Verzeichnis
  - Auto – Motor
  - Mitglied – Verein
  - Zimmer – Haus

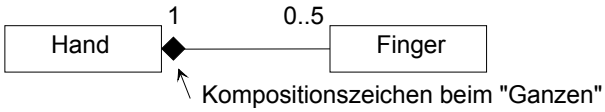
- Arbeitsblatt 4: Polygon (siehe Anhang B.4, entnommen aus [Br01])
- Besprechen des Arbeitsblatts

Komposition, Aggregation

Komposition, Aggregation = Zusammenlagerung, Zusammensetzung

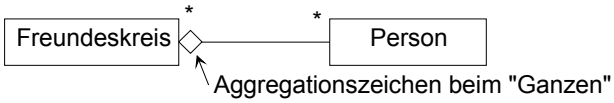
Im objektorientierten Softwareentwurf = Beziehung eines Teils zum Ganzen

Beispiel einer **Komposition** (composite aggregation):



Die Finger sind Teil genau **einer** Hand. In der Regel werden auch die Teile gelöscht, wenn das Ganze stirbt. Hier: Stirbt die Hand, so sterben auch die Finger. Es kann jedoch sein, dass die Finger vor dem Tod der Hand einer anderen Hand zugeordnet werden.
Kompositionen werden häufig verwendet, um einen Erstell- und Löschzusammenhang von Objekten darzustellen.

Beispiel einer **Aggregation** (shared aggregation):



Eine Person ist Teil eines Freundeskreises. Sie kann aber durchaus mehreren Freundeskreisen angehören (-> shared). Aggregationen kommen in der gegenständlichen Welt selten vor.

Aggregationen und Kompositionen werden nur verwendet, wenn auf diesen Zusammenhang Wert gelegt wird. Im Zweifelsfall werden sie weggelassen!

Abbildung 53. Komposition und Aggregation

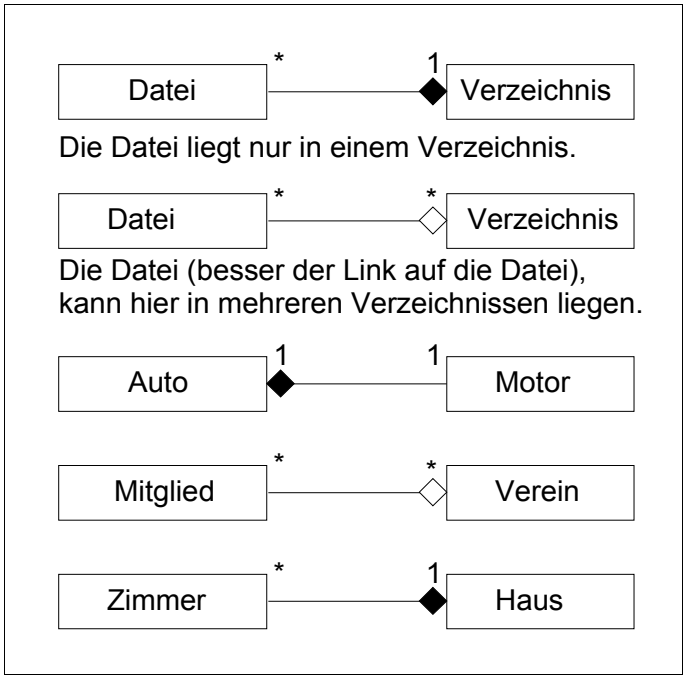


Abbildung 54. Beispiele zur Komposition und Aggregation

A.2.5 Ablauf: Erste Analyse eines größeren Problems (OOA)

- Arbeitsblatt 5 (TicTacToe-Spiel siehe Anhang B.5): Dialog zwischen zwei Personen (Kunde und Softwareexperte), Schüler sollen innerhalb von Kleingruppen in folgender Reihenfolge vorgehen (siehe auch Arbeitsblatt 5):
    1. Klassen suchen
    2. Assoziationen suchen (inkl. Benennung dieser)
    3. Kardinalitäten und Rollen
    4. Attribute

Vorgehen bei der Analyse des Problems

- Suchen nach Klassen (Substantive sind Kandidaten)
- Suchen nach Beziehungen zwischen den gefundenen Klassen
- Suchen nach Kardinalitäten und evtl. nach Rollen
- Suchen nach Attributen in den Klassen (es kann sein, dass für manche Klassen keine Attribute gefunden werden)

Ergebnis: Domain Model, ein Modell aus Sicht des Problembereichs (Domäne des Problems)

Abbildung 55. Vorgehen bei der Problemanalyse

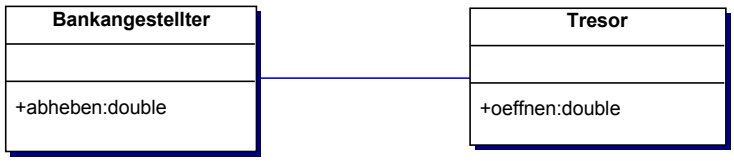
- Danach sollten einige Lösungen exemplarisch besprochen werden, besonders sollte man darauf eingehen, wie die Klassen gefunden wurden (siehe Abbildung 55). Es werden unterschiedliche Lösungen zu finden sein. Wichtig ist, dass es sich hierbei um die Analyse des Problems handelt (OOA), also das WAS im Vordergrund steht. Das WIE muss an dieser Stelle vermieden werden! Das Ergebnis ist im Sinne des Unified Process [La02] [Kr99] kein Klassendiagramm, sondern ein Domain Model, es beschreibt nur den Bereich (die Domäne) des Problems und ist keine Lösung, mit der direkt Softwareklassen erzeugt werden können.
- Arbeitsblatt 6 (siehe Anhang B.6):
  Problem der Klassifizierung [Bc94]. Die Schüler sollen das Problem in Einzelarbeit lösen. Danach sollen sie Ihre Lösungen mit den anderen vergleichen.
  Anschließend gibt es weitere Informationen: Kreise = giftige Chemikalien, Rechtecke = Holz, alle anderen Umrisse = Fahrgäste. Wie ändert diese Information die Klassen?
  → Als Erkenntnis sollte folgen, dass man möglichst viel über die Domäne des Problems wissen sollte. Darin liegt der Sinn der objektorientierten Analyse!

A.2.6 Ablauf: Übergang zum Klassendiagramm mit Hilfe des Sequenz- und Kollaborationsdiagramms

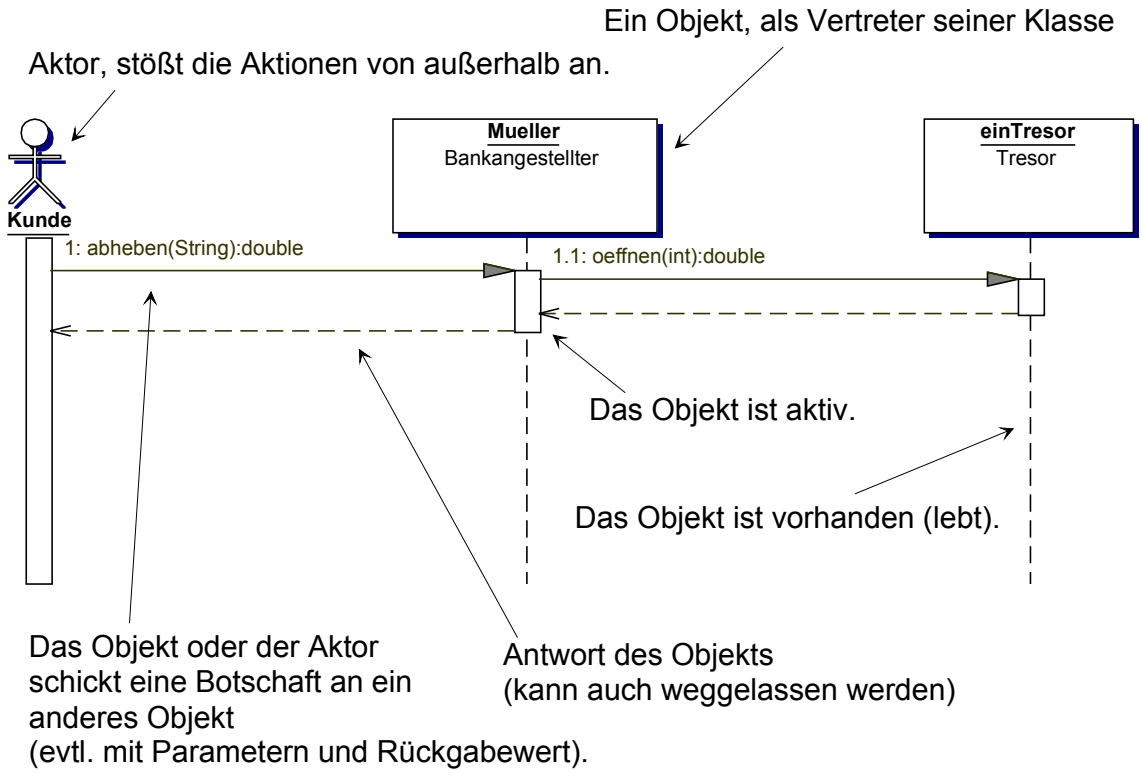
- Das Klassendiagramm zeigt nur den statischen Zustand und die Beziehungen zwischen den Klassen. Wie kann man den zeitlichen Ablauf einer Aktion darstellen?
    - Beispiel des Abhebens an einem Bankschalter (siehe Abbildung 56). Zuerst das Klassendiagramm **ohne** Methoden zeigen. Wie erfolgt das Abheben vom Konto?
    - Wer stößt die Aktion in der Bank an? → der Kunde als Akteur (Aktor)

Das Sequenz- und Kollaborationsdiagramm

Klassendiagramm: Zustand und Beziehungen zwischen den Klassen



Sequenzdiagramm: Ablauf der Aktionen



Das Sequenzdiagramm hilft bei der Suche nach Methoden!

Kollaborationsdiagramm: Zusammenarbeit der Objekte

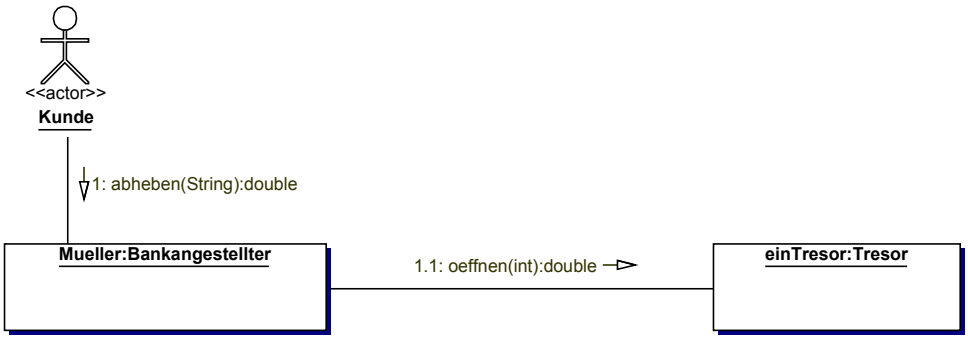


Abbildung 56. Das Sequenz- und Kollaborationsdiagramm.

- An wen wendet sich der Kunde? → Herrn Mueller, ein Objekt der Klasse Bankangestellter
- Wie handelt das Objekt Mueller weiter? → Er wendet sich an das Objekt *einTresor* der Klasse Tresor.
- Welche Methoden müssen die Klassen *Bankangestellter* bzw. *Tresor* unterstützen? → *abheben()* bzw. *oeffnen()*; nun in Klassendiagramm eintragen
- Weitere Darstellungsmöglichkeit der Aktionskette per Kollaborationsdiagramm (Kollaboration = Zusammenarbeiten) zeigen

- Arbeitsblatt 7 (siehe Anhang B.7) :
  Finden von Methoden mit dem Sequenzdiagramm (Gruppenarbeit)
  Exemplarisches Vorstellen der Lösungen durch die Schüler (im Sequenz- und Kollaborationsdiagramm!). Welche Lösung ist die beste? → Designrichtlinien ansprechen (siehe Abbildung 57; GRASP [La02]):
  1. Wer ist der Informationsexperte für diese Aufgabe? D.h. wer hat bzw. sollte die Information für diese Aktion haben, wer muss also angefragt werden?
  2. Geringe Kopplung: Eine Klasse sollte möglichst wenig Verbindungen zu anderen Klassen aufweisen, so sind bei Änderungen weniger Klassen betroffen.
  3. Hoher Zusammenhalt: Die Verantwortlichkeit einer Klasse sollte sich möglichst nur auf eine logische Aufgabe beziehen.

Designrichtlinien

1. Informationsexperte
Wer ist der Informationsexperte für diese Aufgabe? D.h. wer hat bzw. sollte die Information für diese Aktion haben, wer muss also angefragt werden?

2. Geringe Kopplung
Eine Klasse sollte möglichst wenige Verbindungen zu anderen Klassen aufweisen, so sind bei Änderungen weniger Klassen betroffen.

3. Hoher Zusammenhalt
Die Verantwortlichkeit einer Klasse sollte sich möglichst nur auf eine logische Aufgabe beziehen.

4. Erzeuger
Für die Erzeugung neuer Objekte ist die Klasse zuständig, welche die zu erzeugenden Objekte beinhaltet (Komposition, Aggregation) oder die Informationen zum Start des neuen Objekts enthält oder das zu erzeugende Objekt eng verwendet.

Abbildung 57. Die Designrichtlinien (nach [La02])

Nach Designrichtlinie 2 könnte man der Meinung sein, dass alle Methoden in eine Klasse gehören, dann gibt es zu anderen Klassen keine Verbindung. Dies widerspricht jedoch der Richtlinie 3, da dann diese Klasse für alles verantwortlich ist. Hier widersprechen sich also die beiden Richtlinien, was aber als Ergebnis zu vielen in ihren Aufgaben unabhängigen Klassen führt, was das Ziel eines guten Designs darstellt.

- Anschließend kann das Design Model des TicTacToe-Spiels (s.o.) mit Hilfe der Sequenzdiagramme in ein Klassendiagramm umgewandelt werden (siehe Arbeitsblatt 8, Anhang B.8).
  Diese Schritte sind **sehr zeitaufwendig**, weshalb sie als ein Projekt im CT-Unterricht zu Ende gebracht werden sollten. Eine Musterlösung in Form von Diagrammen findet sich im Anhang B.8. Um sich mit dem Thema auseinander zu setzen, wird empfohlen, ohne Verwendung der Musterlösung (welche auch nur eine mögliche Lösung darstellt!) selbst eine Lösung zu finden.
  Bei der Verwirklichung dieses Projekts muss sicher öfter eingegriffen werden, d.h. die bisher entstandenen Designs müssen in der Klasse besprochen werden. Je nach vorhandener Zeit und nach Niveau der Klasse müssen auch einige Codeteile vorgegeben werden. Z.B. malen der Spielsteine, Ermittlung, ob ein Spieler gewonnen hat, ...
  Bei schwachen Klassen oder bei Zeitmangel kann auch der Designprozess im Klassenverband erfolgen und die Codierung entfallen.

A.3 Einheit: Generalisierung

A.3.1 Motivation

Es soll nun weibliche und männliche Kobolde geben. Worin sollen sich in unserem Programm die männlichen Kobolde von den weiblichen unterscheiden?

A.3.2 Ablauf: Generalisierung

- Welche Attribute und Methoden benötigt die Klasse *KoboldMann*? Welche Attribute und Methoden benötigt die Klasse *KoboldFrau*? Wo sind die Gemeinsamkeiten und wo die Unterschiede zwischen den beiden Klassen? (Programm *WillisWelt* v1.11 vorführen, siehe Abbildung 20)
  → Alle Attribute und Eigenschaften sind identisch. Nur die Methode *malen()* ist unterschiedlich, da die Männer einen Bart und die Frauen eine Schleife in den Haaren erhalten. Wenn nun eine Änderung z.B. in der Methode *gehen()* nötig ist, muss diese in 2 Klassen geändert werden. Wie können wir das vermeiden?
  → Wir nehmen alle Methoden und Eigenschaften, die in beiden Klassen identisch sind und fügen sie in die Klasse *Kobold* ein. Die Klasse *KoboldMann* und *KoboldFrau* sind dann Spezialisierungen der Klasse *Kobold* oder umgekehrt ist die Klasse *Kobold* die Verallgemeinerung bzw. die Generalisierung der Klasse *KoboldMann* und *KoboldFrau* (siehe Abbildung 58). Die Methode *malen()* in der Klasse *KoboldFrau* überschreibt die Methode *malen()* seiner Superklasse, es wird die Methode der Subklasse verwendet, deshalb nennt man dies überschreiben.

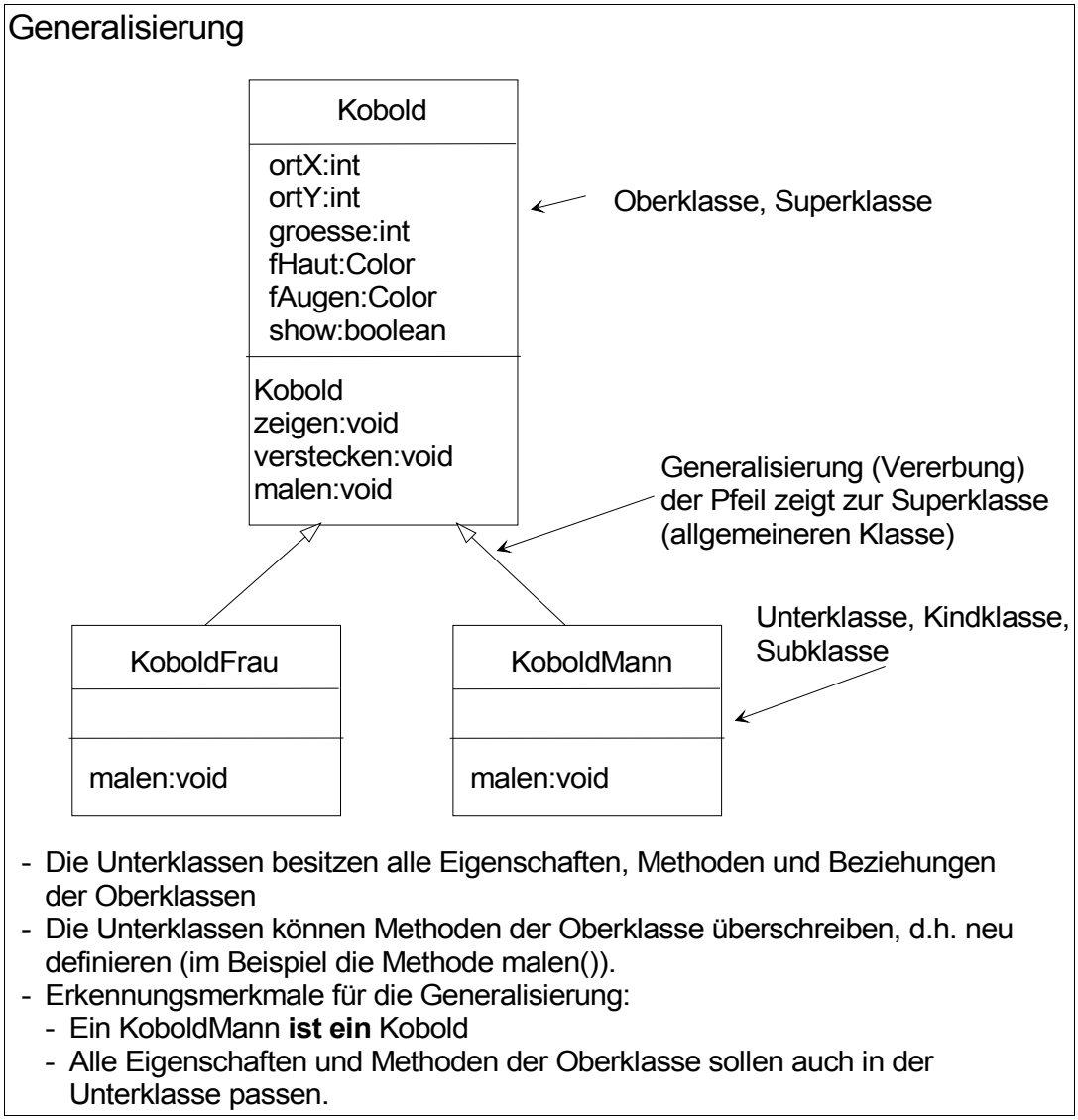


Abbildung 58. Grundlagen zur Generalisierung

- Wie erkennt man, ob generalisiert werden kann?
  → Wenn man sagen kann: Ein KoboldMann **ist ein** Kobold. Weiterhin sollen alle Attribute und Methoden der Superklasse (Oberklasse) zu den Subklassen passen. Weitere Beispiele suchen lassen (Beispiele siehe Abbildung 59). Beim Beispiel Fahrzeug ist auch eine Unterteilung in motorbetrieben und muskelbetrieben möglich. Hier ergibt sich wieder die Diskussion, ob das Modell problemabhängig ist.

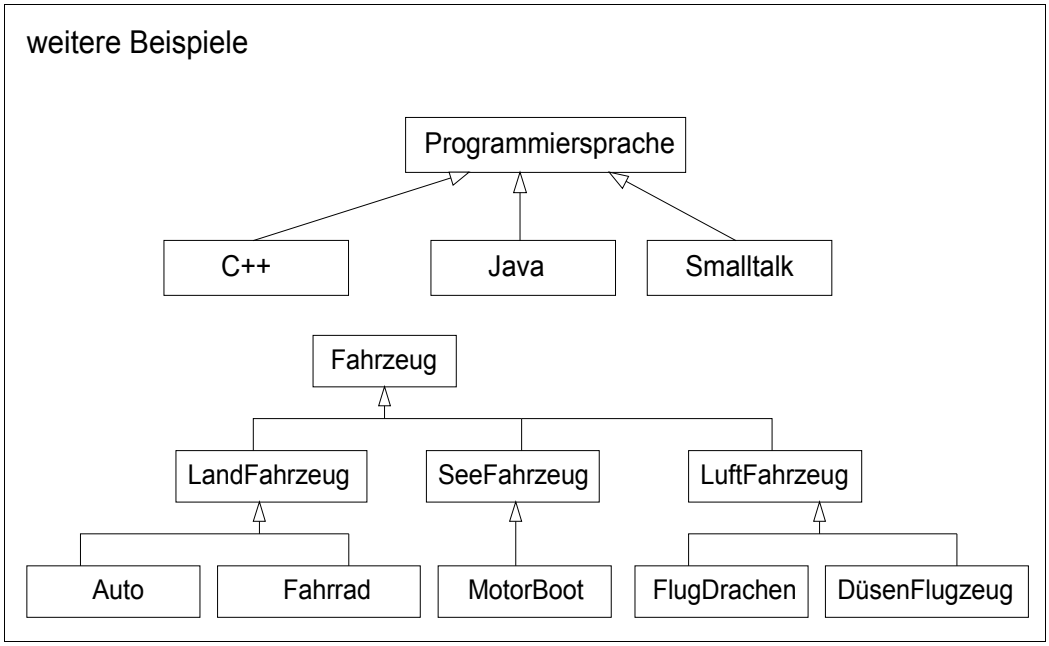


Abbildung 59. Weitere Beispiele zur Generalisierung

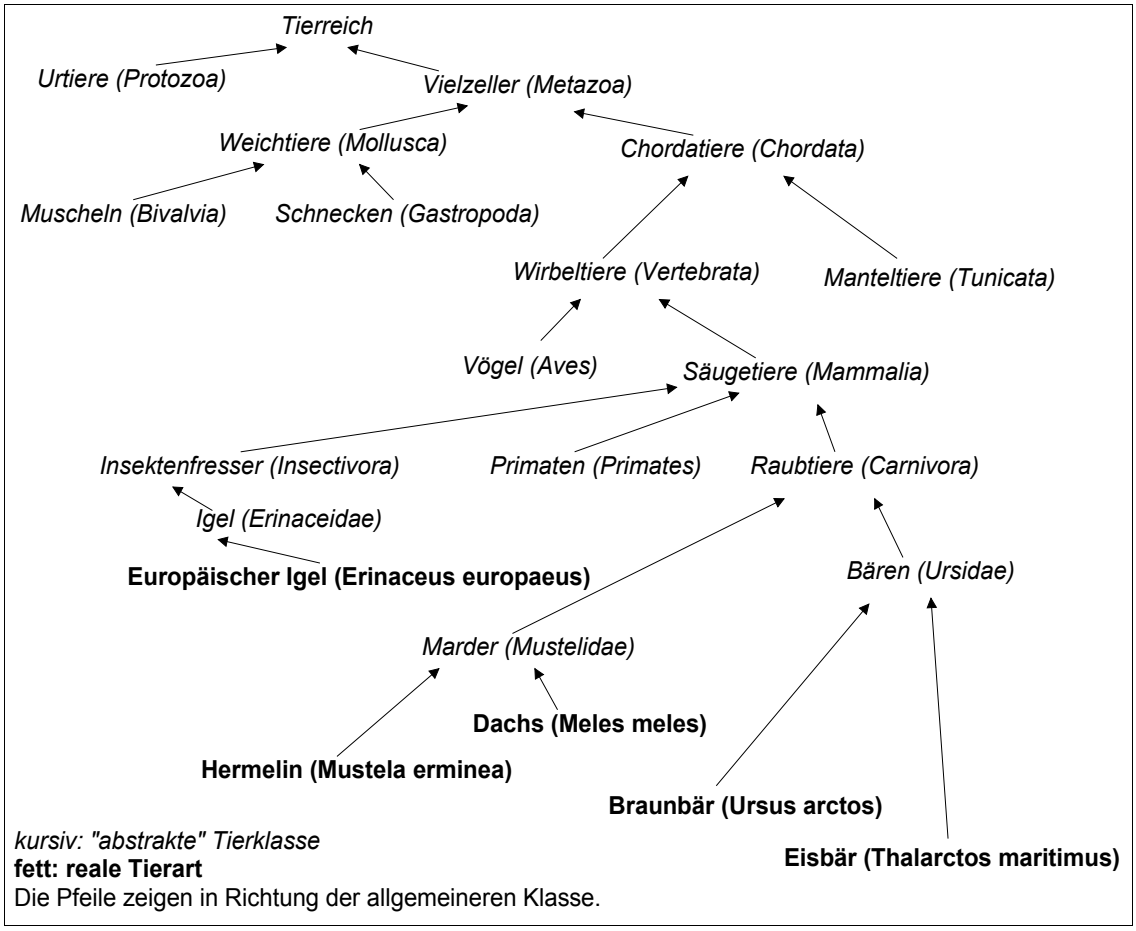


Abbildung 60. Ausschnitt aus dem zoologischen System der Tiere (nach [Za85])

- weiteres Beispiel zur Generalisierung aus einem anderen Fachbereich: zoologisches System der Tiere (siehe Abbildung 60).

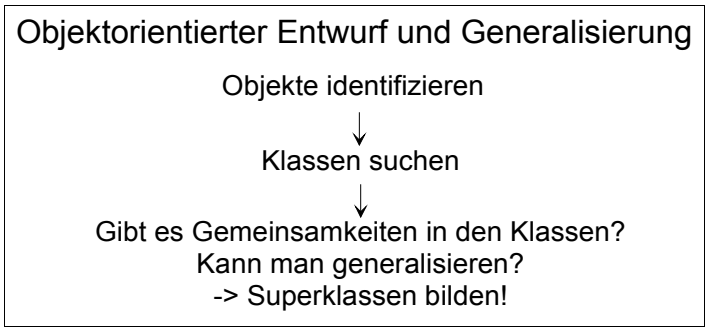


Abbildung 61. Vorgehen zum Klassenentwurf unter Verwendung der Generalisierung [Wi]

- Vorgehen beim Klassenentwurf, wenn Generalisierung verwendet werden soll: Bild mit Bildern, Worten, Piktogrammen → Klassendiagramm suchen. (Vorgehen, siehe Abbildung 61; Aufgabe siehe Arbeitsblatt 9, Anhang B.9) [Wi]

- Wie sieht die Generalisierung im Code aus?
  → evtl. die Schüler die Generalisierung mit einem CASE-Tool erzeugen lassen (Programmcode in Java siehe Abbildung 62), dann den Code untersuchen, die beiden Methoden zum Malen der Kobolde vorgeben

Vererbung in Java

LandFahrzeug ist Superklasse von Auto (to extend: dt. erweitern)

```
class Auto extends LandFahrzeug {
   ...
   //Konstruktor
   Auto(int fahrgestellnummer) {
      super(fahrgestellnummer);//Konstruktor von
                               //LandFahrzeug wird aufgerufen
      ...
   }
   ...
}
```

Soll der Konstruktor der Superklasse aufgerufen werden, muss dies der *erste* Befehl im Konstruktor sein.

Abbildung 62. Die Vererbung in der Programmiersprache Java

- Welche Fehlermeldungen gibt es beim Compilieren?
  → Die privaten Eigenschaften in der Klasse *Kobold* verursachen eine Fehlermeldung. Das Zugriffsrecht *protected* einführen, Verwenden des Konstruktors der Oberklasse (Java: *super()*; C++: Initialisierungsliste des Konstruktors). (für Java siehe Abbildung 62 und 63)

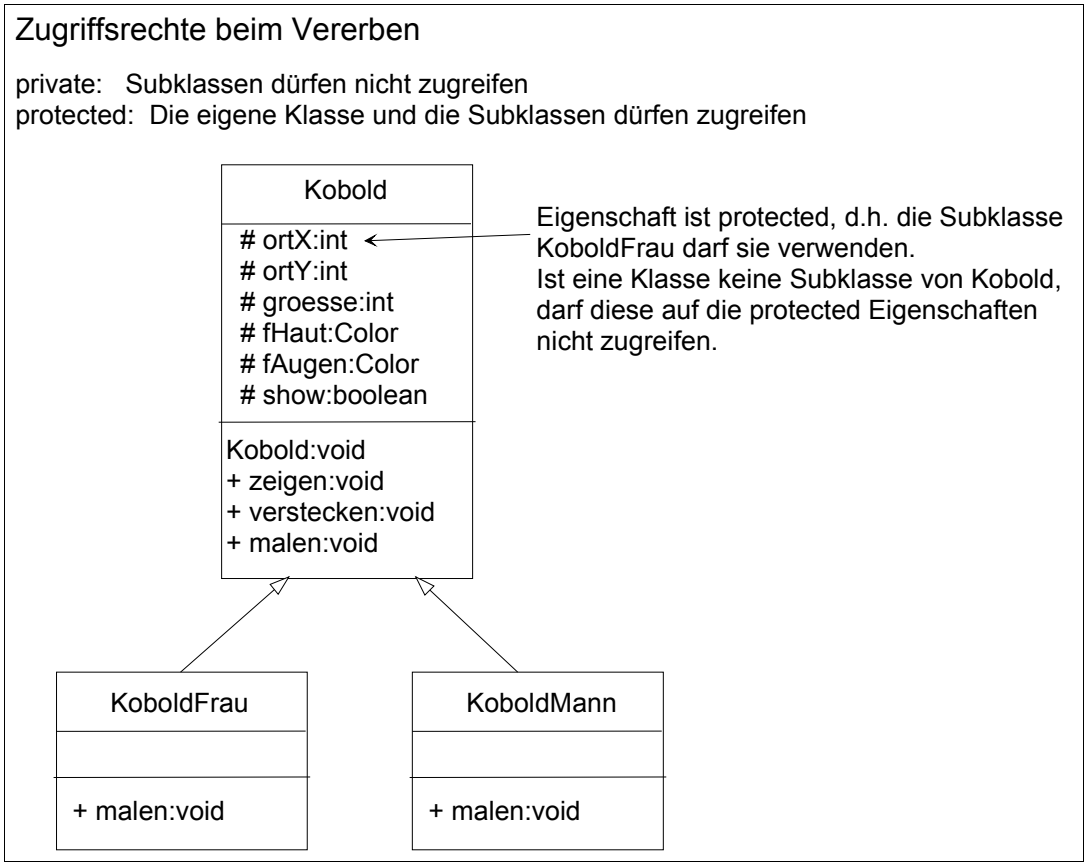


Abbildung 63. Die Zugriffsrechte beim Vererben

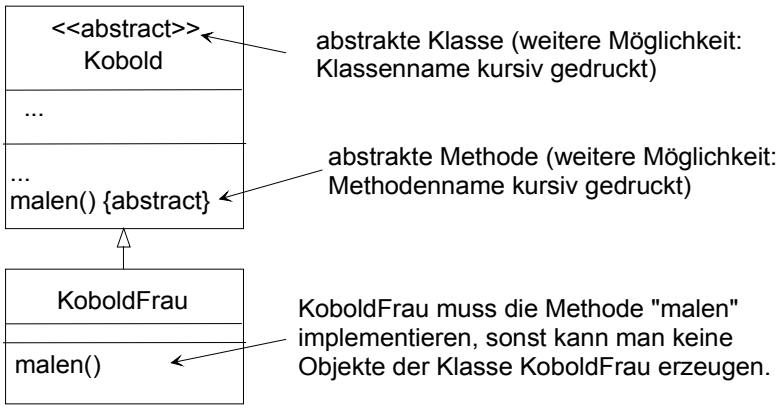
Für C++ muss die Klasse noch virtuell werden (Schlüsselwort : virtual, sonst werden nicht die Methoden der Kinder verwendet), in Java sind alle Klassen immer virtuell. Beim Programmieren nennt man die Generalisierung vererben, da die Unterklassen die Eigenschaften und Methoden der Oberklassen erben.
→ Programmieren von *KoboldFrau* und *KoboldMann* (Hinweis: Im Konstruktor der Klasse *Krone* wird nun bereits Polymorphie verwendet)

- Wird die Methode *malen()* in der Klasse Kobold noch benötigt? Soll es möglich sein, dass ein Objekt der Klasse *Kobold* erzeugt wird?
  → Wenn der Programmierer der Klasse *Kobold* möchte, dass alle Subklassen die Methode *malen()* neu schreiben **müssen** (überschreiben) und niemand ein Objekt der Klasse *Kobold* erzeugen darf, kann er sie als abstrakte Klasse mit der abstrakten Methode *malen()* anfertigen. Abstrakt heißt hier, dass es keine reale Klasse bzw. Methode ist, da sie nicht direkt verwendbar sind, d.h. es kann kein Objekt dieser Klasse erzeugt werden.
  → Im Programm die Klasse *Kobold* und ihre Methode *malen()* als abstrakt deklarieren (In C++ erreicht man dies durch rein virtuelle Methoden, siehe WillisWelt v1.11)

Abstrakte Methoden und Klassen

Soll es von einer Superklasse keine Objekte geben, sondern nur von den Subklassen, so kann man sie als "abstrakt" deklarieren.
In abstrakten Klassen kann es auch abstrakte Methoden geben. Diese haben keine Implementierung. Eine Subklasse muss eine Implementierung für diese Methode besitzen, sonst ist auch sie eine abstrakte Klasse.

Beispiel:



In Java

```
public abstract class Kobold {
      ...
      public abstract void malen();
      ...
}
```

Abbildung 64. Abstrakte Klassen und Methoden

Hat eine Klasse eine abstrakte Methode, muss auch die Klasse abstrakt sein! (siehe Abbildung 64)
In Abbildung 60 (Ausschnitt zoologisches System) sind alle Knoten abstrakt, nur die Blätter sind Klassen, von denen reale Tiere existieren.

A.3.3 Ablauf: Klassen und Vererbungen finden

- Ein Programm zum Zeichnen von geometrischen Figuren (Kreise, Rechtecke, Dreiecke) soll geschrieben werden. Weiterhin soll die Gesamtfläche aller geometrischen Figuren berechnet werden können (siehe Abbildung 65).

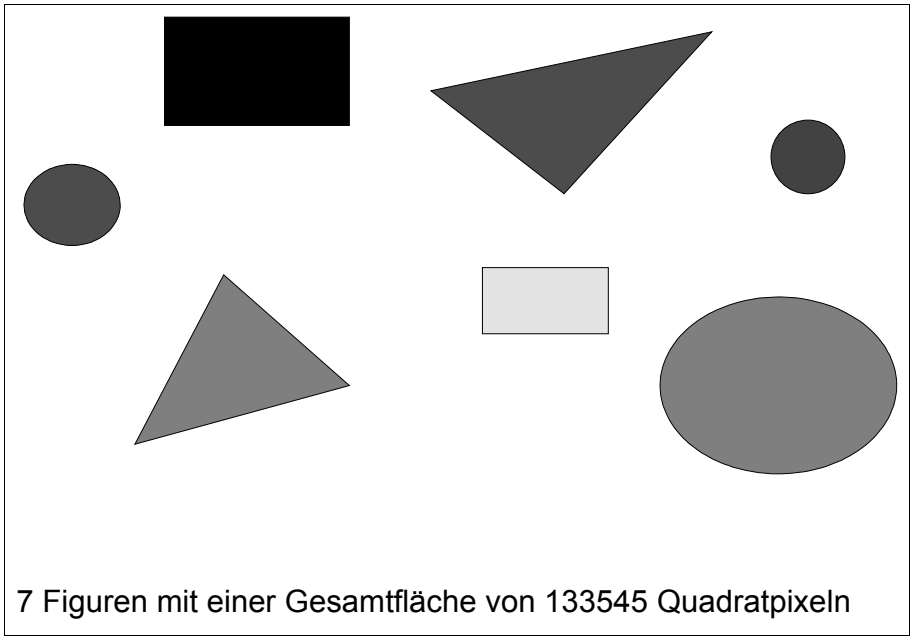


Abbildung 65. Die Problemstellung zum Graphik-Projekt

- Klassendiagramm der Problemdomäne (Domain Model) von Schülern entwerfen lassen. Das Vorgehen entspricht dem Vorgehen in der vorangegangenen Untereinheit. Es handelt sich hierbei um eine objektorientierte Analyse (OOA) → Lösung siehe Abbildung 66. Die Klasse zum Anzeigen der Figuren (*GraphikFrame*) und die Klassen für die Dialoge zum Zeichnen der Figuren sind erst das Ergebnis des Designs!
  Vorerst wird mit einem statischen Array von 20 Elementen gearbeitet.

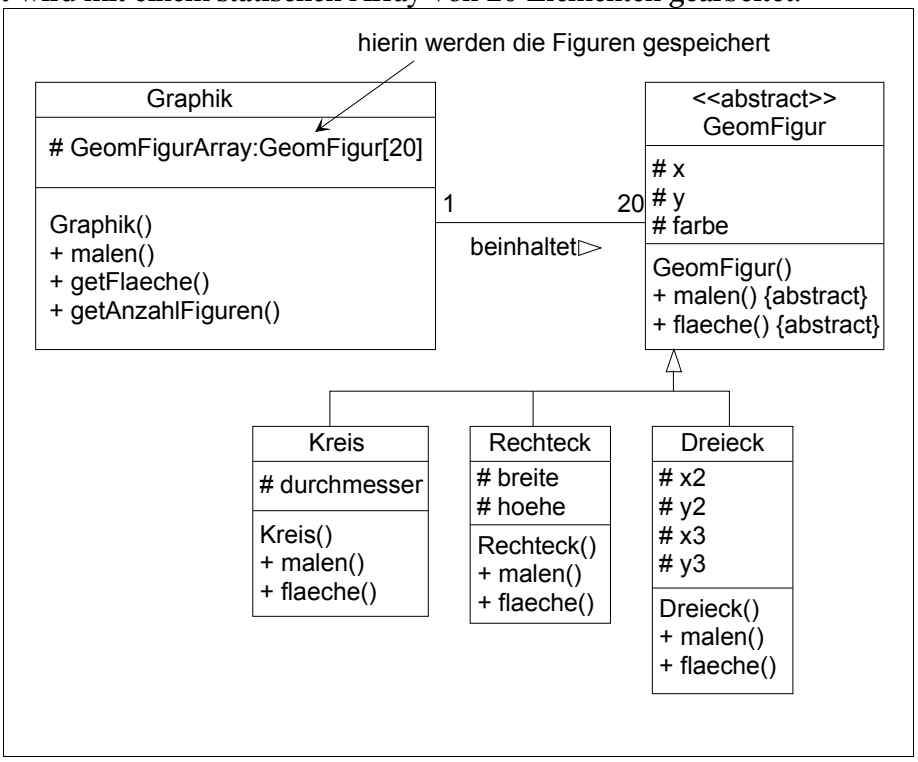


Abbildung 66. Das Analysediagramm (Domain Model) zum Graphik-Projekt

- Eine wichtige Frage beim Übergang zum Design ist: Wie speichern wir die einzelnen Graphikobjekte? Sollen wir für jedes Objekt ein eigenes Array erzeugen? Nachteil: Unser Bild hat viele Dreiecke, sodass dieses Array evtl. voll ist, es hat aber noch viel Platz für Rechtecke.
  Lösung: Ein Array mit Objekten der gemeinsamen Oberklasse. Dann sind alle Methoden der Oberklasse zugänglich (genauer: die Schnittstellen). Da ein Objekt aber selbst weiß, ob es ein Dreieck oder ein Rechteck ist, verwendet es die dann richtige Implementierung der Methode. Dies nennt man Vielgestaltigkeit (Polymorphie), da vor einem Aufruf der Methode nicht bekannt ist, welche Implementierung verwendet wird. Die Methode hat viele Gestalten → siehe Abbildung 67

Polymorphie
poly: dt. viel
Morphologie: dt. Gestalt
Polymorphie = Vielgestaltigkeit

Wird in der Klasse *Graphik* eine Figur aus dem Array *GeomFigurArray* herausgeholt und gezeichnet, ist nicht bekannt, ob es sich um einen Kreis, ein Rechteck oder ein Dreieck handelt. Erst zur Laufzeit ist klar, um welche Figur es sich handelt.
Das Figuren-Objekt weiß "selbst", ob es z.B. ein Kreis ist, und wählt die richtige Methode zum Malen eines Kreises.
Bei der Polymorphie erscheint eine Methode in vielen "Gestalten".

Da im Array nur der Datentyp *GeomFigur* gespeichert wird, sind auch nur dessen Methodennamen zugänglich. Deshalb müssen die Methoden *flaeche* und *malen* in *GeomFigur* deklariert sein.
Zur Laufzeit wird dann aber die passende Methode der Subklasse verwendet.

Abbildung 67. Die Polymorphie

- Die Schüler sollen zuerst die Klassen *GeomFigur*, *Kreis*, *Rechteck* und *Dreieck* implementieren. In der Klasse *Graphik* soll zum Testen zuerst ein Array der Klasse *Kreis*, bzw. der Klasse *Rechteck* oder *Dreieck* verwendet werden. Wenn dies funktioniert, soll ein Array der Klasse *Graphik* angelegt werden.
  Lösung siehe Graphik v1.1
- Wir möchten uns nun nicht auf 20 Figuren beschränken. Wie kann man eine (fast) unbeschränkte Anzahl von Figuren speichern, ohne vorher zu wissen, wie viele es werden? → dynamisches Array, ein Container (Java siehe Arbeitsblatt 10, Anhang B.11; in der Standard Template Library STL [ISO98] von C++ gibt es entsprechende Klassen, ältere C++-Compiler bieten ähnliche Bibliotheken, siehe auch den Beispielcode in Graphik v1.3). Ändern der Implementierung mit einem Container. Wenn das Klassendiagramm durchdacht ist, muss man nur in der Klasse *Graphik* Änderungen vornehmen, denn nur sie ist für die Verwaltung der Objekte zuständig! Die Schnittstelle ändert sich nicht (nur der Konstruktor der Klasse kann vereinfacht werden: nun ist kein Übergabeparameter mehr nötig). → Graphik v1.3
- Ein weiterer Button zum Löschen aller Objekte soll hinzugefügt werden. Wie muss man das Klassendiagramm ändern? Am Klassendiagramm besprechen (Hinzufügen der entsprechenden Methoden). → Graphik v1.5

A.4 Einheit: Weitere Entwurfsmethoden

A.4.1 Motivation zum Zustandsdiagramm

Wir sollen ein Zahlenschloss bauen wie es an manchen Häusern anstatt eines Schlüssels zu finden ist. Wie gehen wir vor? Wie ein Automat (Spielautomat) reagiert die Maschine abhängig von der Eingabe und den vorangegangenen Ereignissen. Bei einem Spiel ist z.B. der weitere Spielverlauf vom Spiel-Level abhängig.

A.4.2 Ablauf: Zustandsdiagramm und CVS

- Wie kann man das Verhalten des Zahlenschlosses beschreiben? → verbale Beschreibung des Verhaltens durch die Schüler. Betonen, dass die Maschine sich merken muss, dass bereits die erste bzw. erste und zweite Zahl richtig eingegeben wurde. Das Verhalten der Maschine ist davon abhängig, was zuvor eingegeben wurde → Die Maschine hat unterschiedliche Zustände!
- Wie kann man das Verhalten/die Zustände des Zahlenschlosses graphisch darstellen? → Zustandsdiagramm einführen; Startzustand, Ereignis und zuletzt die Aktion (siehe Abbildung 68)
  Weitere Beispiele für Zustandsautomaten von den Schüler bearbeiten lassen: Alarmanlage, Paritäts-Erkennung (siehe Arbeitsblatt 11, Anhang B.12)

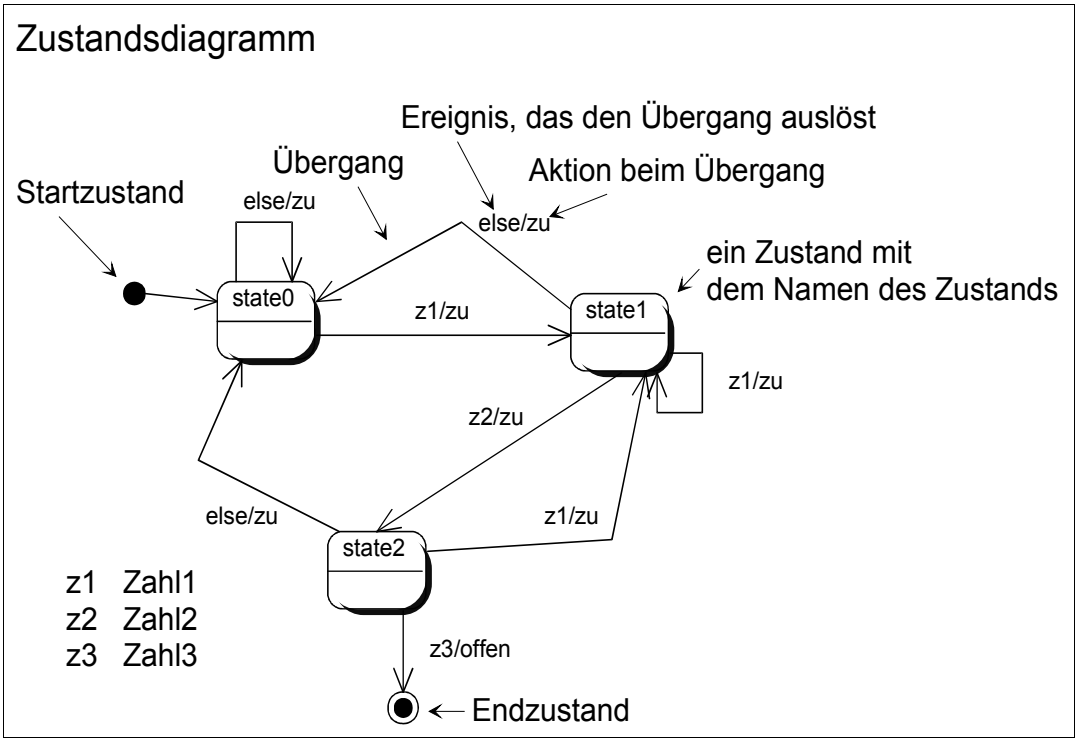


Abbildung 68. Das Zustandsdiagramm für das Zahlenschloss

- Wie soll sich der Automat nach außen hin verhalten (als Black Box)? → Wie muss die entsprechende Klasse aussehen (siehe Abbildung 69)? Die Methode *ereignis()* kann den Aktionswert zurückliefern, es ist aber auch möglich, eine eigene Methode *getAktion()* zu implementieren, die den Aktionswert liefert.

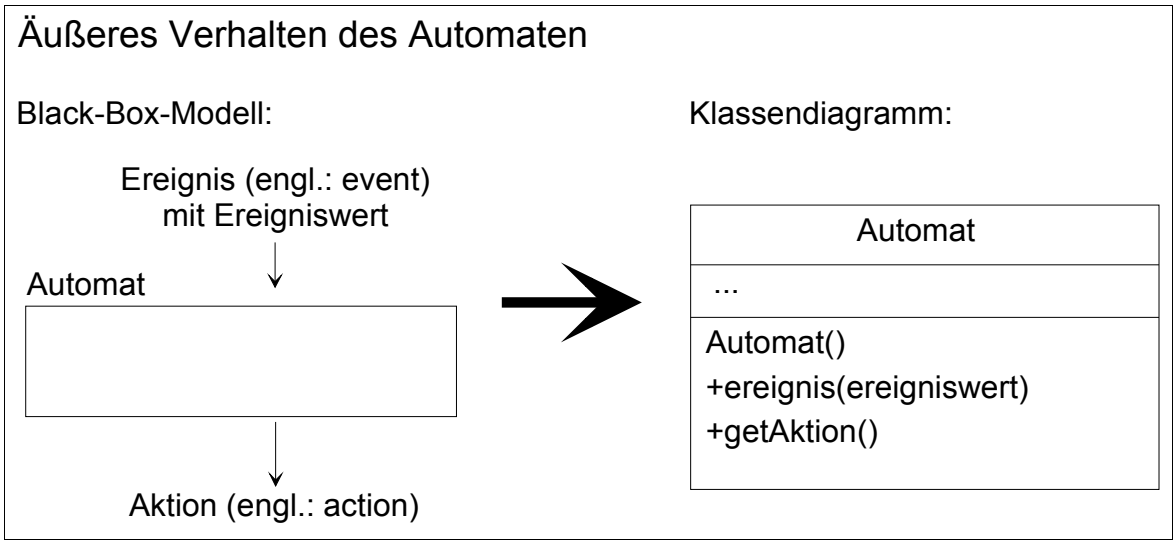
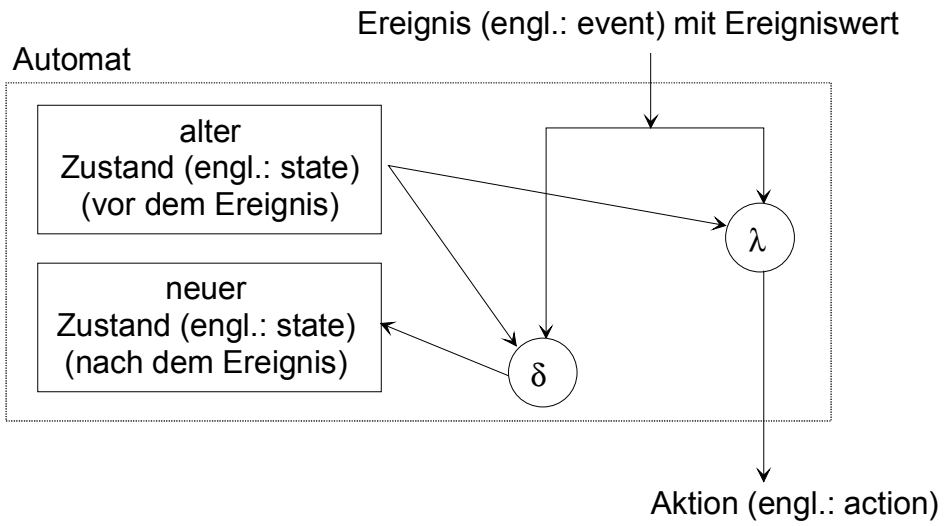


Abbildung 69. Der Automat als Black-Box und das passende Klassendiagramm

- Wie soll sich der Automat im Innern verhalten? Wie programmiert man einen Zustandsautomat? Internes Modell des Automaten vorstellen, Übergangstabelle δ und Ausgangstabelle λ aus dem Zustandsdiagramm entwickeln (siehe Abbildung 70)
- Wie kann man mit Hilfe der Tabellen den Automat programmieren?
  → In Kleingruppen Entwurf des Struktogramms für den allgemeinen tabellengesteuerten Automaten. Vorstellung und Besprechung der Ergebnisse. Je nach Niveau der Schulklasse kann der Algorithmus auch zuvor verbal mit der Klasse in einem Unterrichtsgespräch entwickelt werden.
  Im zweiten Schritt Entwurf des Struktogramms des Zahlenschlosses (Erweiterung des Struktogramms des allg. Automaten, v.a. des Algorithmus zur Zuordnung der Token; siehe Abbildung 71)

Modell des Automaten



Übergangsrelation δ, ändert den internen Zustand:

		Ereignisse			
		z1	z2	z3	else
Zustände	state0	-> state1	-> state0	-> state0	-> state0
	state1	-> state1	-> state2	-> state0	-> state0
	state2	-> state1	-> state0	-> Ende	-> state0

Ausgangsrelation λ, ändert die Ausgabe:

		Ereignisse			
		z1	z2	z3	else
Zustände	state0	zu	zu	zu	zu
	state1	zu	zu	zu	zu
	state2	zu	zu	offen	zu

Über eine Festlegung wird nun jedem Zustand, jedem Ereignis (bzw. Zeichen, engl. token) und jeder Aktion eine Zahl zugeordnet. Z.B:

state0 -> 0, state1 -> 1, state2 -> 2
z1 -> 0, z2 -> 1, z3 -> 2, else -> 3
zu -> 0, offen -> 1

Mit dieser Festlegung kann man die beiden Tabellen als 2-Dim-Array realisieren.

Abbildung 70. Inneres Modell des Automaten, die Übergangsrelation und die Ausgangsrelation

Allgemeiner Automat

ereignis(ereigniswert):ganzzahl

Verarbeiten eines neuen Ereignisses des Automaten. Zurückgegeben wird die Aktion.
ausgabeTabelle -> 2-Dim-Array mit der Ausgangsrelation (Attribut)
uebergangsTabelle -> 2-Dim-Array mit der Übergangsrelation (Attribut)
aktion und zustand sind Attribute der Klasse

Token ermitteln
aktion := ausgabeTabelle[zustand][token]
zustand := uebergangsTabelle[zustand][token]
Rückgabe: aktion

Zahlenschloss

ereignis(ereigniswert):ganzzahl

Verarbeiten eines neuen Ereignisses des Zahlenschlosses. Zurückgegeben wird die Aktion.
code[] -> Array mit den 3 Zahlen des Zahlenschlosses (Attribut)
ausgabeTabelle -> 2-Dim-Array mit der Ausgangsrelation (Attribut)
uebergangsTabelle -> 2-Dim-Array mit der Übergangsrelation (Attribut)
aktion und zustand sind Attribute der Klasse

ereigniswert ==			
code[0]	code[1]	code[2]	sonst
token := 0	token := 1	token := 2	token := 3
aktion := ausgabeTabelle[zustand][token]			
zustand := uebergangsTabelle[zustand][token]			
Rückgabe: aktion			

Abbildung 71. Das Struktogramm für den allgemeinen Automaten und das Zahlenschloss

- Wie soll die Klasse *Frame* zur Anzeige und zur Verarbeitung der Eingaben aussehen? Trennung der Funktionalität der Anwendung (Controller) und der Anzeige (View)! Dadurch wird es einfacher, das Programm später z.B. als Applet zu portieren, nur die Klasse Frame muss geändert werden (siehe Abbildung 72).
  → Programmieren des Problems (→ Zahlenschloss-v1.1)
- Weiteres Beispiel: Texterkennung; in einem Text nach allen vorkommenden Formen des Familiennamens Maier suchen (regulärer Ausdruck (Kleinschreibung): m[ae][iy]er), (siehe Arbeitsblatt 12, Anhang B.13; Software siehe maier v1.1). In CT entwickeln und programmieren

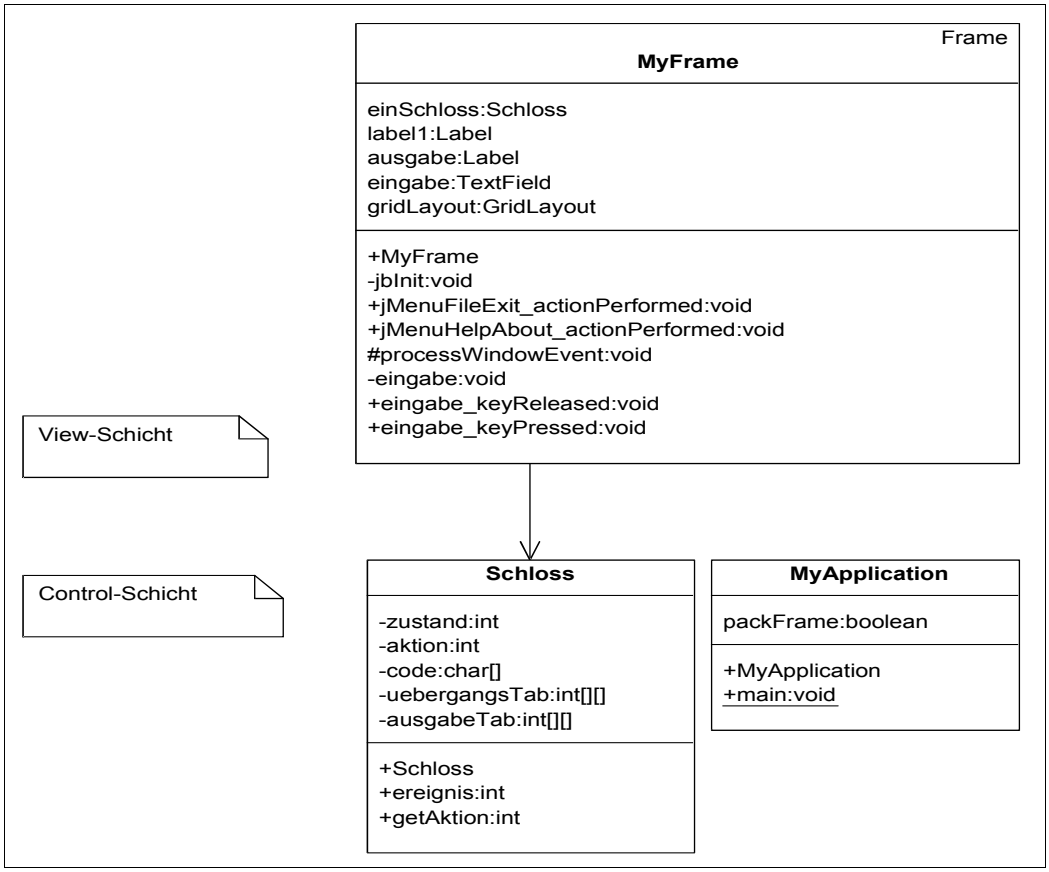


Abbildung 72. Das Klassendiagramm zum Zahlenschloss mit View- und Control-Schicht

- Während des Betriebs stürzt der Rechner ab. Wie können wir erreichen, dass unser Automat den momentanen Zustand nicht vergisst?
  → abspeichern auf der Festplatte oder in einer Datenbank.
  Wo realisieren wir diese Funktionalität?
  → eine eigene Klasse verwenden, so muss man sich in den anderen Klassen nicht festlegen, wo und wie die Daten gesichert werden → Trennung der Anwendung in Control, View, Storage (CVS) in Schichten (siehe Abbildung 73). Auf Arbeitsblatt 12 (Maier-Suchmaschine) kann das Programm entsprechend erweitert werden (siehe Software maier v1.1)

- die Klasse *Speicher* als Code verteilen, die Schüler sollen sie in ihr Programm einbinden
  → *Zahlenschloss v1.3*
- in Computertechnik die Maier-Suchmaschine so erweitern, dass der Text aus einer Datei geladen wird (siehe Software maier v1.1)

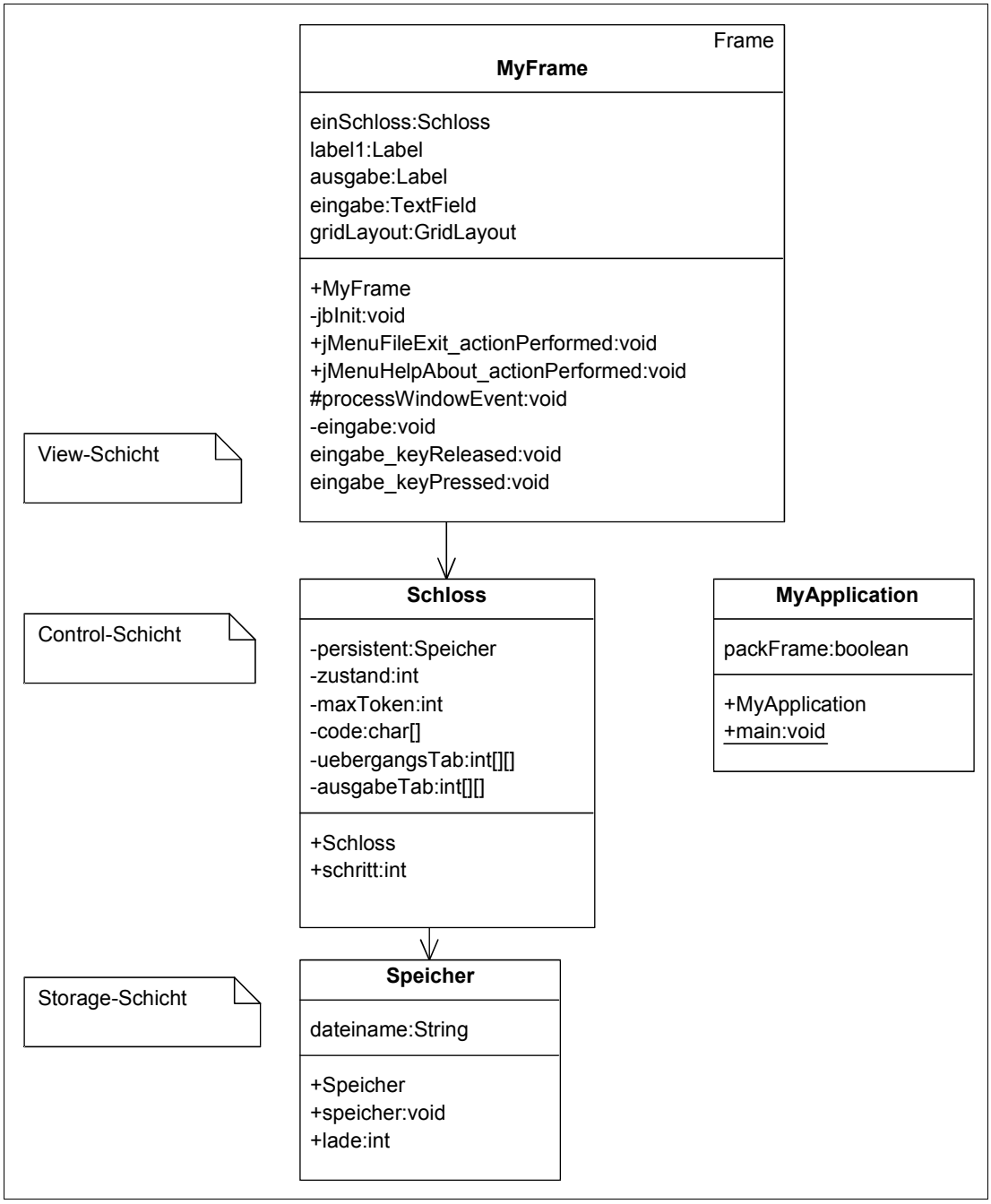


Abbildung 73. Das Klassendiagramm zum Zahlenschloss mit dem CVS-Pattern

A.4.3 Motivation zum Anwendungsfall (Use Case)

Wir sollen eine Registrierkasse programmieren. Wie gehen wir vor, um das Problem zu verstehen? Das WAS ist hier wichtig (OOA). Es handelt sich um ein komplexes Problem, weitere Informationen haben wir noch nicht. Wie gehen wir an dieses Problem heran, so dass wir weitere Informationen für dieses Projekt bekommen?

A.4.4 Ablauf: Anwendungsfall (Use Case)

- Wer muss später als Anwender mit der Registrierkasse arbeiten und könnte als Fachmann gefragt werden?

→ Ein Kaufmann bzw. ein Kassierer kennt sich in der Regel mit dem Ablauf an der Kasse aus. Aber auch alle anderen Beteiligten (im Unified Process: Stakeholder (Kr99]) sind wichtig und sollten gefragt werden (z.B. ein Kunde, der einkauft). Nur so erhält der Softwareexperte ein umfassendes Bild über die Aufgaben der Software.

- Wie geht man bei der Befragung des Kassierers am besten vor?
  - Wer ist bei den Vorgängen mit der Registrierkasse im weitesten Sinne beteiligt?
    → Akteure finden
  - Welche Prozesse stoßen die Akteure an?
    → Anwendungsfälle (Use Cases) finden

  Das Use-Case-Diagramm wird dabei Stück für Stück entwickelt (eine mögliche Lösung zeigt Abbildung 74). Wenn die Use Cases gefunden wurden, kann man Prioritäten vergeben (z.B. unabdingbar, wichtig, optional). Use Cases mit hoher Priorität werden dann zuerst weiter untersucht und später auch zuerst implementiert.
- danach das Vorgehen zusammenfassen

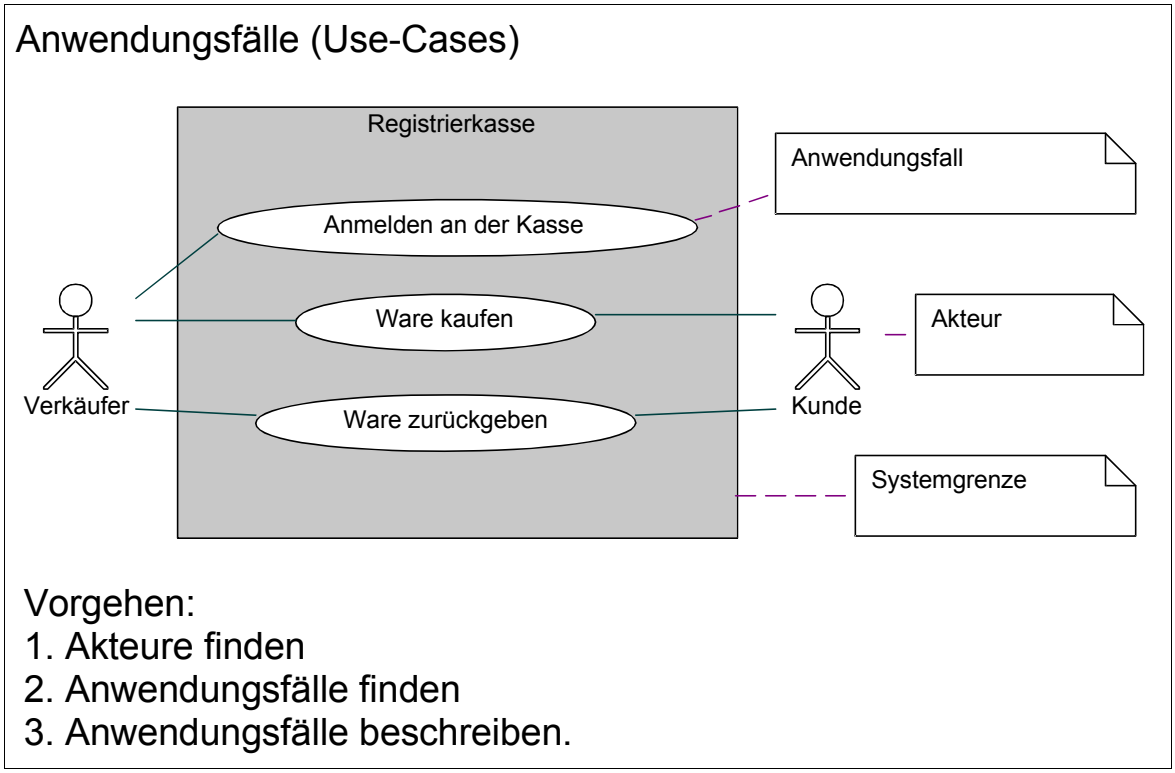


Abbildung 74. Anwendungsfälle (Use Cases) und das Vorgehen zum Finden dieser

- Nun wird der Fachmann für jeden Anwendungsfall nach einem typischen Ablauf (Szenario) gefragt. Dabei erstellt man eine strukturierte Beschreibung des Ablaufs (siehe Abbildung 75). Die Lösung sollte dabei mit den Schülern erarbeitet werden und wird wahrscheinlich nicht der angegebenen entsprechen. Wichtig ist die Struktur (Anwendungsfall, Akteur, Kurzbeschreibung, typischer Ablauf mit Unterscheidung der Aktion des Akteurs und der Antwort des Systems) und das stückweise Vorgehen bei der Entwicklung des Szenarios.
  Zuerst nur den typischen Ablauf behandeln.

Anwendungsfall: **Ware kaufen**
Akteure: Kunde (anstoßend), Verkäufer
Beschreibung: Ein Kunde erscheint an der Kasse mit der Ware, die er kaufen möchte. Der Verkäufer gibt die Waren in die Kasse ein und ermittelt den Gesamtbetrag. Der Kunde zahlt den Gesamtbetrag und nimmt die Ware mit.
Typischer Ablauf:

Aktion des Akteurs	Antwort des Systems
1. Der Kunde erscheint an der Kasse.	
2. Der Verkäufer gibt die Warennummer in die Kasse ein.	
	3. Die Kasse gibt den Einzelpreis aus.
4. Der Verkäufer gibt die Menge in die Kasse ein.	
	5. Die Kasse gibt die Summe dieses Postens aus.
6. 2. bis 5. kann sich wiederholen.	
7. Der Verkäufer schließt die Eingabe ab.	
	8. Die Kasse gibt die Endsumme aus.
9. Der Kunde zahlt.	
10. Der Verkäufer gibt den erhaltenen Betrag des Kunden in die Kasse ein.	
	11. Die Kasse ermittelt das Rückgeld.
12. Der Verkäufer gibt dem Kunden das Rückgeld.	
13. Der Kunde verlässt den Laden.	

Alternativen:
- Bei Schritt 6 entscheidet sich der Kunde, diese Ware nicht zu nehmen bzw. in einer anderen Menge.
- Bei Schritt 9 bemerkt der Kunde, dass er nicht genügend Geld hat → Abbruch des Vorgangs.

Benutzeroberfläche (User Interface, UI):

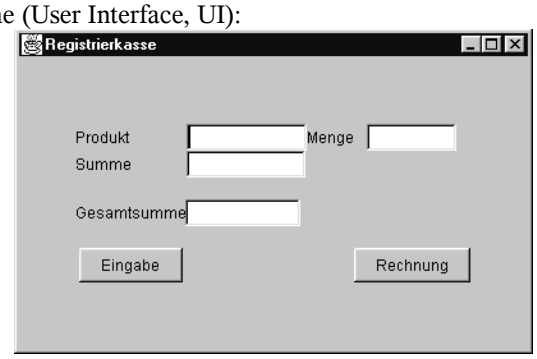


Abbildung 75. Strukturierte Beschreibung des Anwendungsfalls "Ware kaufen"

- Welche Alternativen zum typischen Ablauf gibt es?
  → Nach dem typischen Ablauf beschreiben (siehe Abbildung 75)
- Später kann der Use Case mit der dazu passenden Benutzeroberfläche ergänzt werden.
- Zur Übung sollen die Schüler in Kleingruppen zu weiteren Use Cases die Beschreibung anfertigen. Evtl. können von verschiedenen Gruppen unterschiedliche Anwendungsfälle beschrieben werden → die Lösungen dann von den Kleingruppen vorstellen und diskutieren lassen. Zur Erstellung und zur Vorstellung eignet sich ein Textverarbeitungsprogramm o.ä. (sofern eine Projektionsmöglichkeit, z.B. Beamer vorhanden ist).

- Aus dieser Beschreibung kann man dann das Domain Modell nach den bisher gelernten Verfahren entwerfen. In der Regel sind aber die Use Cases sehr komplex, sodass eine Umsetzung sehr aufwendig und zeitintensiv ist! Der Hinweis sollte hier jedoch gemacht werden.
- Als weitere Übung (Hausaufgabe) kann Arbeitsblatt 13 (Anhang B.14) verwendet werden (evtl. nur einen Use Case genauer beschreiben lassen). Hier kann man die Schüler anregen, eine Vorlage für ein Textverarbeitungssystem zu erstellen.
  Bei der Besprechung wird sich zeigen, dass es viele verschiedene Lösungen gibt.

A.4.5 Überblick über den Entwurfsprozess

- Wie geht man allgemein vor, wenn ein neues Programm entworfen werden soll?
  → Diskussion mit den Schülern über den gesamten bisher behandelten Stoff; arbeiten an der Metaplanwand! Aufschreiben der Schritte durch die Schüler, dann in eine logische Reihenfolge bringen lassen! Abbildung 76 zeigt als Überblick ein mögliches Ergebnis. Der iterative Ansatz wird durch den Lehrer ergänzt.
  Wichtig ist der iterative Ansatz und die Unterscheidung der unterschiedlichen Schritte (Use Case, OOA, OOD, OOP).

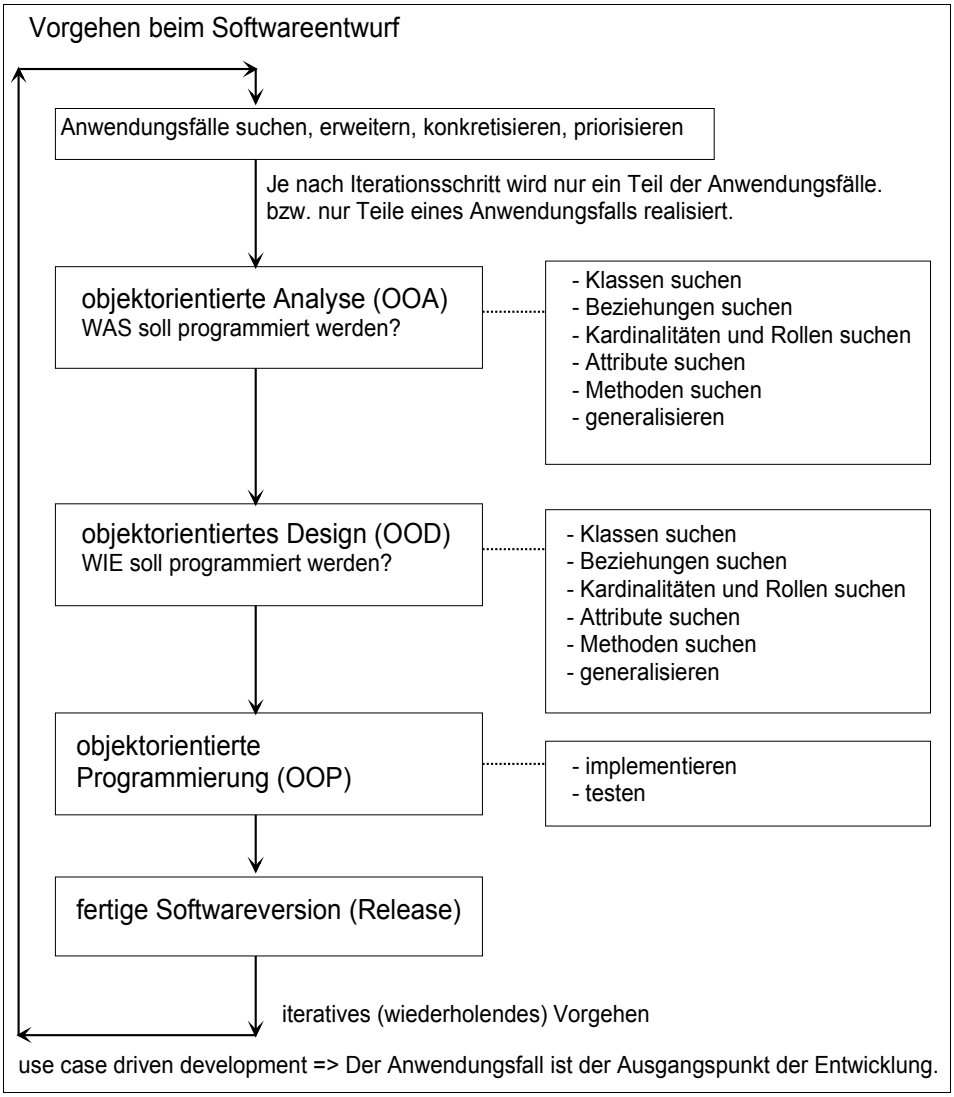
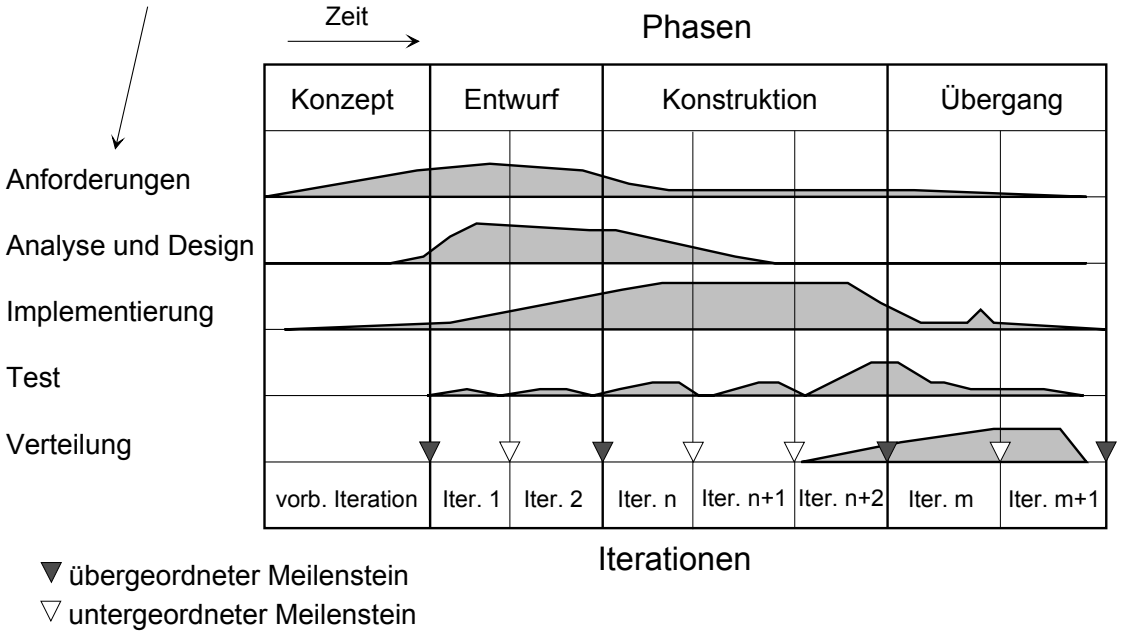


Abbildung 76. Der iterative Prozess beim objektorientierten Softwareentwurf

Grobplanung, Phasenplan

Arbeitsabläufe (engl.: work flow) mit ihrer Gewichtung während des Gesamtprozesses



Die einzelnen Iterationen werden erst kurz vorher geplant!

Phase	Beschreibung
Konzept	- Das Projektziel festlegen (sehr grob)
Entwurf	- Use Case finden - Projektplan (grober Zeit- und Resourcenplan) - Grobziele festlegen - Grobe Architektur festlegen (Domain Modell) - Gibt es problematische Teile im Projekt, dann entsprechende Studien durchführen!
Konstruktion	- Domain Model verfeinern - Design Modell entwerfen - dynamische Modelle (Zustandsdiagramme, Sequenz- und Kollaborationsdiagramme) - Implementierung (Alpha-Release) - Benutzerdokumentation
Übergang	- Vorbereitung auf Verteilung (Verkauf o.ä.) der Software - Fine-Tuning der Anwendung (Beta-Release) - Benutzerschulung

Arbeitsablauf	Beschreibung
Anforderung	- Anforderungen des Problems suchen, z.B. mit Anwendungsfällen
Analyse und Design	- Problem analysieren (Ergebnis: Domain Modell) und - eine Lösung entwerfen (Ergebnis: Design Modell)
Implementierung	- Programmcode erzeugen
Test	- Programmcode testen, ob Anforderungen erfüllt werden
Verteilung	- Benutzer schulen, die Anwendung „fein tunen“, etc.

Dokumentiert wird in allen Phasen und Arbeitsabläufen!

Abbildung 77. Der vereinfachte Unified Process (nach [Kr99]).

- Wie werden die einzelnen Iterationen angeordnet? Ist in jeder Iteration jeder Schritt gleichbedeutend?
  Der Unified Process [Kr99] wird kurz und vereinfacht vorgestellt. So wird ein grobes Vorgehen in der Gesamtplanung aufgezeigt. In Abbildung 77 (vereinfacht nach [Kr99]) werden die 4 Phasen des Unified Process (UP) dargestellt und die einzelnen Aufgabenbereiche (eigentlich Workflows, die im UP detailliert beschrieben werden) im Ablauf qualitativ gewichtet. Dieser Teil erfolgt stark lehrerzentriert und ist bereits die Vorbereitung für ein nachfolgendes Projekt, in welchem die Schüler die Prozessschritte konkret nachvollziehen sollen.

B Arbeitsblätter mit Lösungen

Die zum Teil hier vorgestellten Lösungen sind nicht als die einzig Richtigen zu verstehen. Oft stellen sie nur eine Lösung unter vielen dar. Häufig sind, vor allem bei den offener gestellten Aufgaben, von den Schülern einfachere Lösungen zu erwarten.

B.1 Arbeitsblatt 1: Assoziationen

1. Assoziationen

Zeichnen Sie die Klassendiagramme (inkl. der Kardinalitäten und der Benennung der Assoziationen) für folgende Beispiele:

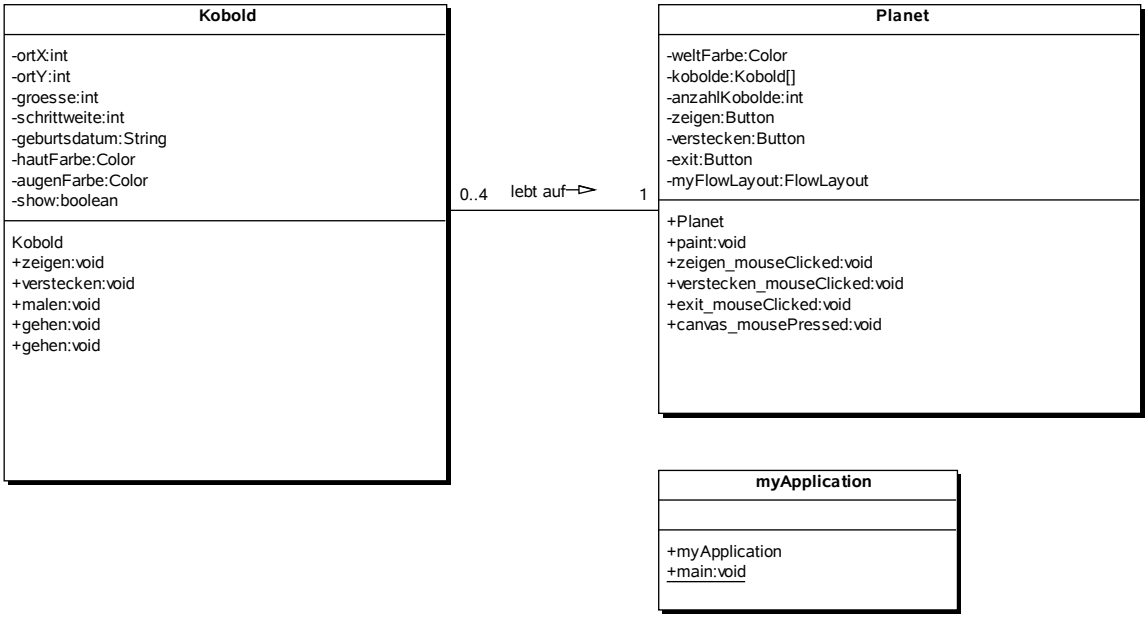
a) Auto – Reifen
b) Buch – Seite
c) Auto – Strasse
d) Baum – Blätter
e) Walkman – Kopfhörer
f) Dirigent – Orchestermusiker
g) Trainer – Spieler
h) Orchester – Musiker

2. Assoziationen im Code

Finden Sie anhand des Listings des Programms *Willis Welt* (auf dem PC) heraus, wie die Assoziation zwischen den Kobolden und dem Planeten in der Programmiersprache realisiert ist. Schreiben Sie hierzu die wichtigsten Zeilen heraus!

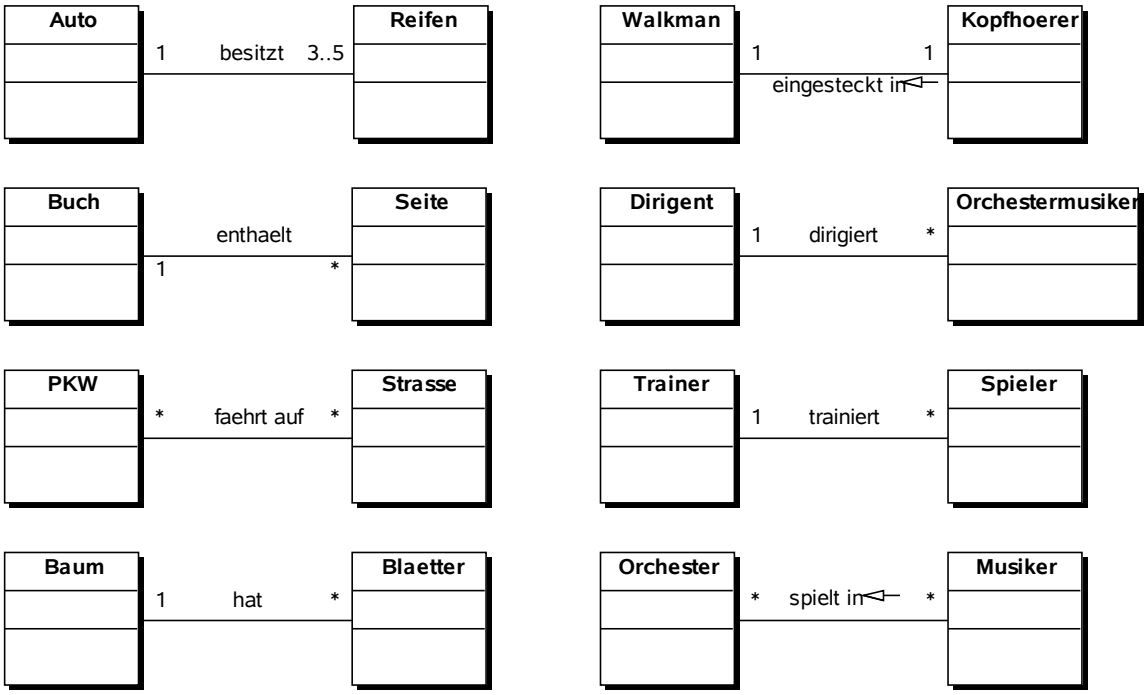
3. Größere Kardinalitäten im Code

Überlegen Sie, wie man die Kardinalität 0..4 im Programmcode realisieren kann.
Hinweis: Array und Zähler für Anzahl der Elemente
Erweitern Sie Ihr bisheriges Programm so, dass das unten stehende Klassendiagramm realisiert wird.
Hinweis: *zeigen()*, *verstecken()*, *malen()* und die Bewegung beim Anklicken der Zeichenfläche soll sich immer auf alle Kobolde beziehen.



B.1.1 Mögliche Lösung zum Arbeitsblatt 1: Assoziationen

1. Assoziationen



2. Assoziationen im Code

```
public class Planet extends Frame {

      //Die Attribute der Klasse
      private Kobold willi; //dieses Attribut stellt die Assoziation her
      ...
      //Die Methoden der Klasse
      //Konstruktor, wird beim Erzeugen der Klasse aufgerufen
      public Planet(Color wFarbe) {
          ...
          //die beiden Kobolde werden geboren
          willi = new Kobold(250, 100, 50, Color.red,   Color.blue, 20);
          ...
          }
      }
```

3. Größere Kardinalitäten im Code

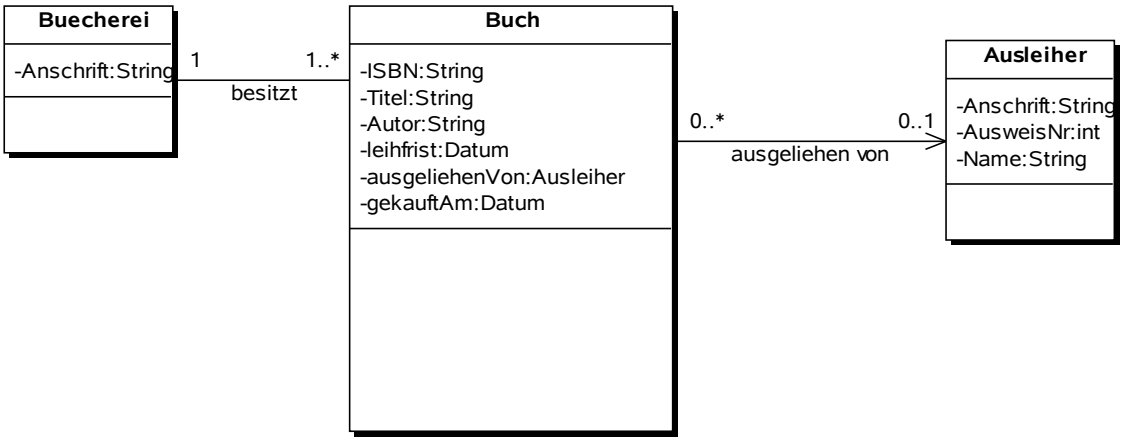
siehe Programmcode zu WillisWelt v1.7.

B.2 Arbeitsblatt 2: Assoziationen und OOA 1

Ein Klassendiagramm für ein einfaches Ausleihsystem einer Bücherei soll 3 Klassen erhalten: Bücherei, Buch, Ausleiher.

1. Zeichnen Sie die Assoziationen ein und benennen Sie diese.
2. Geben Sie alle Kardinalitäten an.
3. Ergänzen Sie die Klassen um geeignete Attribute.

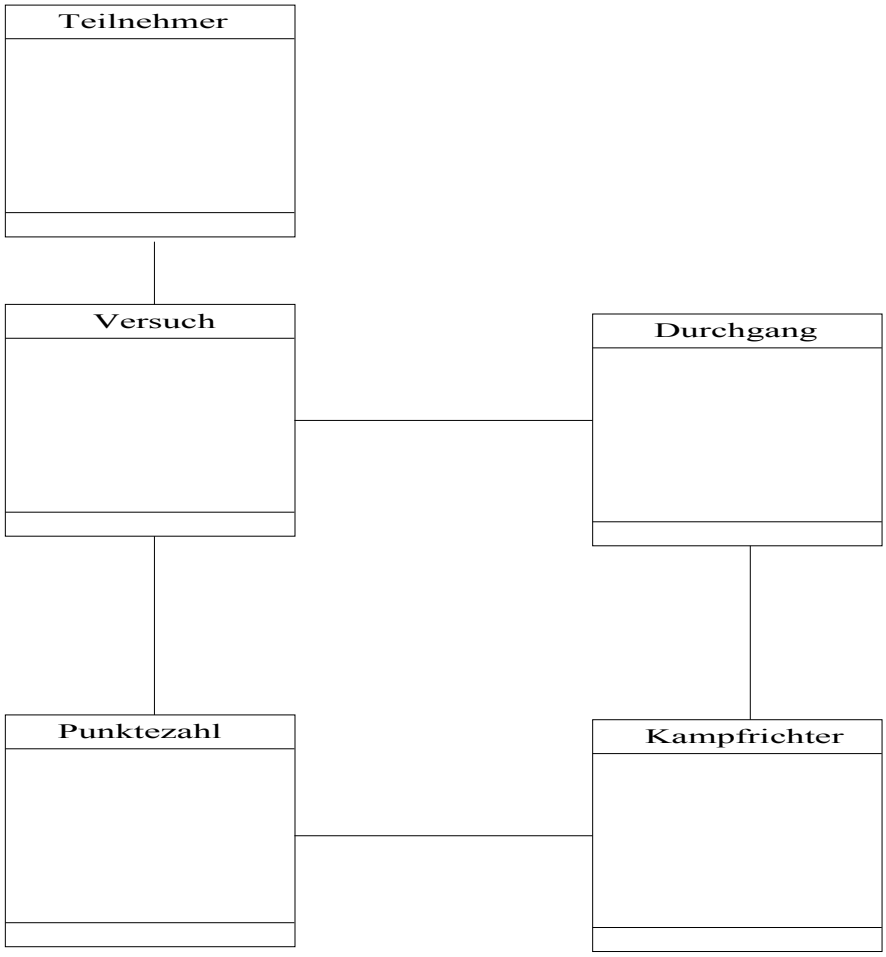
B.2.1 Mögliche Lösung zum Arbeitsblatt 2: Assoziationen und OOA



B.3 Arbeitsblatt 3: Assoziationen und OOA 2

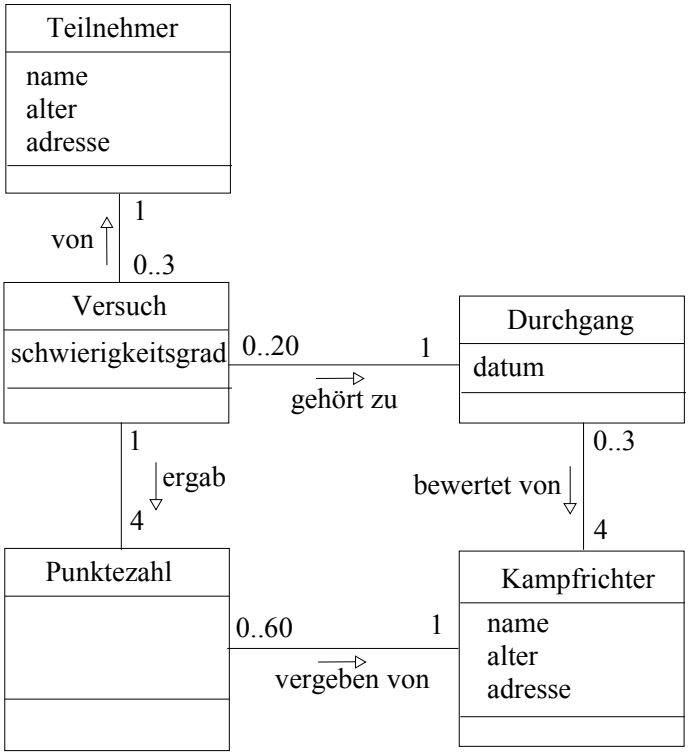
Unten ist ein Klassendiagramm, das beim System zur Vereinfachung der Planung und Bewertung von Sportwettkämpfen wie Gymnastik, Kunstspringen und Eiskunstlauf eingesetzt werden könnte. Dabei gibt es mehrere Durchgänge und Wettbewerbsteilnehmer. Ein Durchgang ist abgeschlossen, wenn alle Teilnehmer einen Versuch abgelegt haben.
Jeder Teilnehmer tritt in 3 Durchgängen an und für jeden Durchgang gibt es 20 Teilnehmer.
Für jeden Durchgang gibt es 4 Kampfrichter, welche die Leistung der Teilnehmer in diesem Durchgang subjektiv bewerten. Es kann vorkommen, dass ein Kampfrichter nicht alle Durchgänge bewertet. Er wird dann durch einen anderen ersetzt. Während eines Durchgangs wird aber kein Kampfrichter ersetzt.
Jeder Kampfrichter bewertet jeden Teilnehmer in einem Durchgang. In der Regel punktet ein Kampfrichter mehr als einen Durchgang.
Jeder Versuch ist ein Start eines Teilnehmers, bei einem Durchgang die bestmögliche Leistung zu erbringen. Ein Versuch wird durch eine Jury von Richtern für diesen Durchgang gepunktet und die Endpunktzahl wird ermittelt.
Es soll nachher nachvollziehbar sein, welcher Kampfrichter bei welchem Versuch (Start) wie viel Punkte vergeben hat.

1. Fügen Sie den Assoziationen Namen und Kardinalitäten hinzu.
2. Fügen sie folgende Attribute hinzu: adresse, alter, datum, schwierigkeitsgrad, name. In einigen Fällen können diese in verschiedenen Klassen vorkommen.



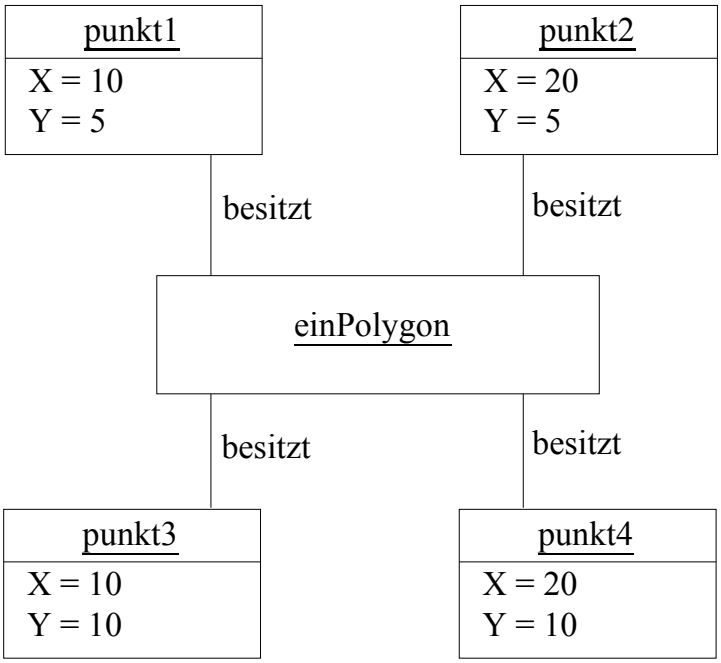
(nach [Br01], variiert)

B.3.1 Mögliche Lösung zum Arbeitsblatt 3: Assoziationen und OOA



B.4 Arbeitsblatt 4: Aggregation/Komposition

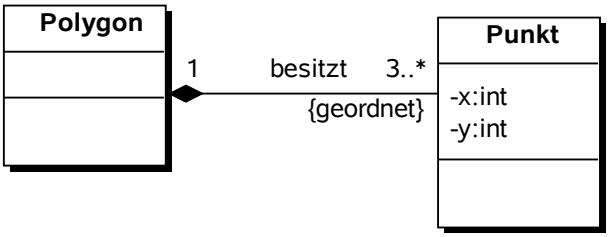
Folgendes Objektdiagramm ist gegeben:



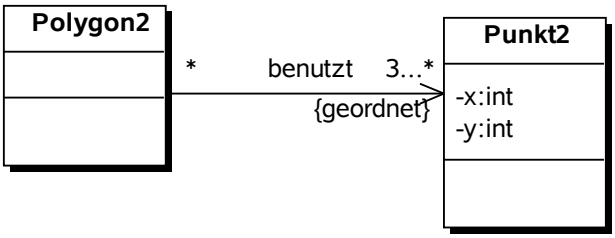
(nach [Br01])

1. Wie könnte das Klassendiagramm aussehen?
2. Kann man mehrere Lösungen finden?
3. Wie unterscheiden sich diese?

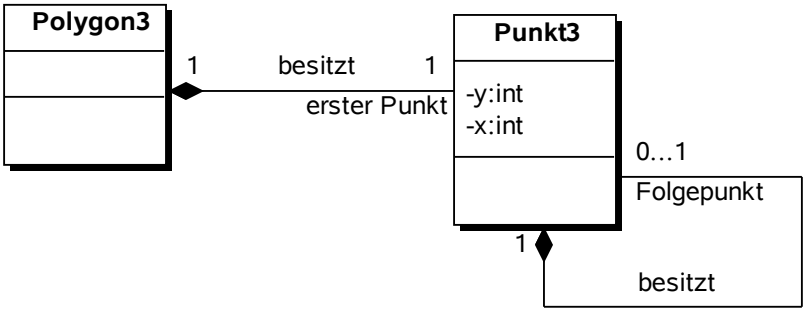
B.4.1 Mögliche Lösungen zum Arbeitsblatt 4: Aggregation/Komposition



Hier gehört jeder Punkt zu genau einem Polygon. Wird das Polygon gelöscht, so werden auch alle Punkte gelöscht. Der Zusatz in geschweifter Klammer (*{geordnet}*) ist Teil der UML und gibt an, dass die Abfolge der Punkte geordnet ist.



Hier können mehrere Polygone den selben Punkt verwenden. Ein Punkt ist nicht mehr Teil eines Polygons, sondern gehört zu mehreren Polygonen. Dies spart evtl. Speicherplatz, ist aber in der Realisierung recht schwierig.



Hier gehört der Startpunkt zum Polygon und jeder weitere Punkt zum Vorgänger des Punktes. Dies entspricht einer verketteten Liste. Es wird eine reflexive Assoziation verwendet.
Wird das Polygon gelöscht, so wird auch der erste Punkt gelöscht. Wird dieser gelöscht, so wird auch sein Nachfolger gelöscht und so weiter.

B.5 Arbeitsblatt 5: Erste Problemanalyse (OOA)

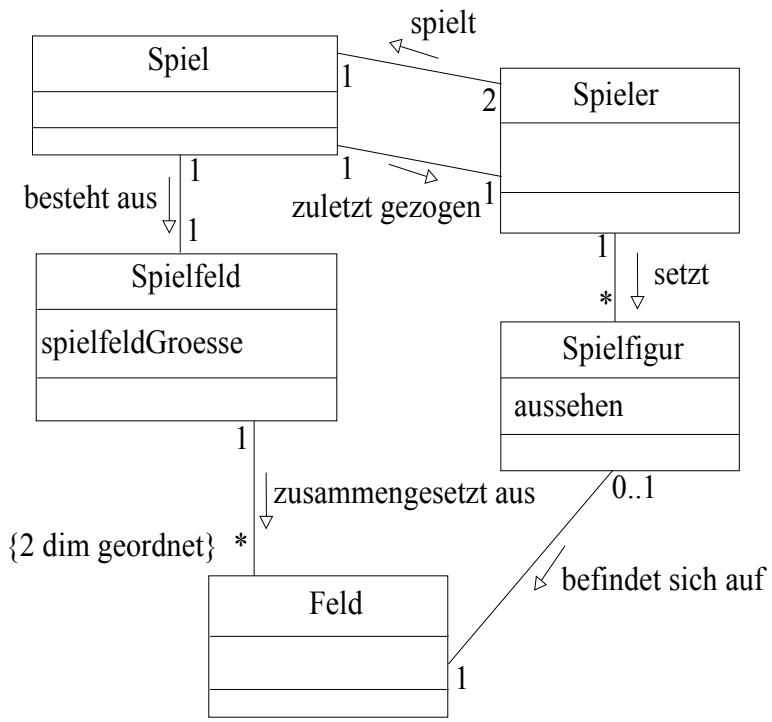
Für einen Kunden soll Software entworfen werden. Bei einem ersten Treffen findet folgendes Gespräch statt, dabei ist K der Kunde und S der Softwareentwickler:

K: Ich möchte, dass Sie ein Spiel für mich programmieren.
S: Erklären Sie, wie das Spiel funktioniert.
K: Das Spielfeld besteht aus 3x3 Feldern, kann aber auch größer sein.
S: Ist das Feld dann immer quadratisch?
K: Ja, es kann auch aus 4x4 oder 5x5 Feldern bestehen usw.
S: Wie viele Spieler nehmen an diesem Spiel teil?
K: Bei diesem Spiel spielen 2 Spieler gegeneinander. Jeder hat seine Spielfiguren, z.B. der eine ein Kreuz, der andere einen Kreis.
S: Können es auch andere Zeichen sein?
K: Ja, man kann z.B. auch unterschiedliche Farben verwenden. Die Spielfiguren der beiden Spieler müssen sich nur unterscheiden.
S: Was ist das Ziel des Spiels?
K: Ziel ist es, dass ein Spieler alle Felder in einer Reihe, einer Spalte oder in einer Diagonalen mit seinen Spielfiguren belegt hat. Hat er das geschafft, so hat er gewonnen.
S: Was ist, wenn keiner der beiden Spieler das erreicht?
K: Dann wird so lange gespielt, bis alle Felder belegt sind.
S: Erklären Sie nun den Spielablauf.
K: Der Spieler der anfängt – sagen wir, Spieler 1 – setzt eine seiner Spielfiguren auf ein beliebiges freies Feld.
S: Wie wird entschieden, wer anfängt?
K: Das wird ausgelost.
S: Wie geht es dann weiter?
K: Dann setzt Spieler 2 eine seiner Spielfiguren auf ein beliebiges, freies Feld. Dabei sollte die Figur so gesetzt werden, dass die Gewinnchancen des Gegners verschlechtert und die Eigenen verbessert werden.
Danach kommt wieder Spieler 1, der genauso weitermacht. Man spielt also immer abwechselnd.
S: Wie lange geht es so weiter?
K: So lange, bis ein Spieler gewonnen hat, also alle Felder in einer Reihe, einer Spalte oder einer Diagonalen belegt hat oder alle Felder belegt sind.

Aufgaben:

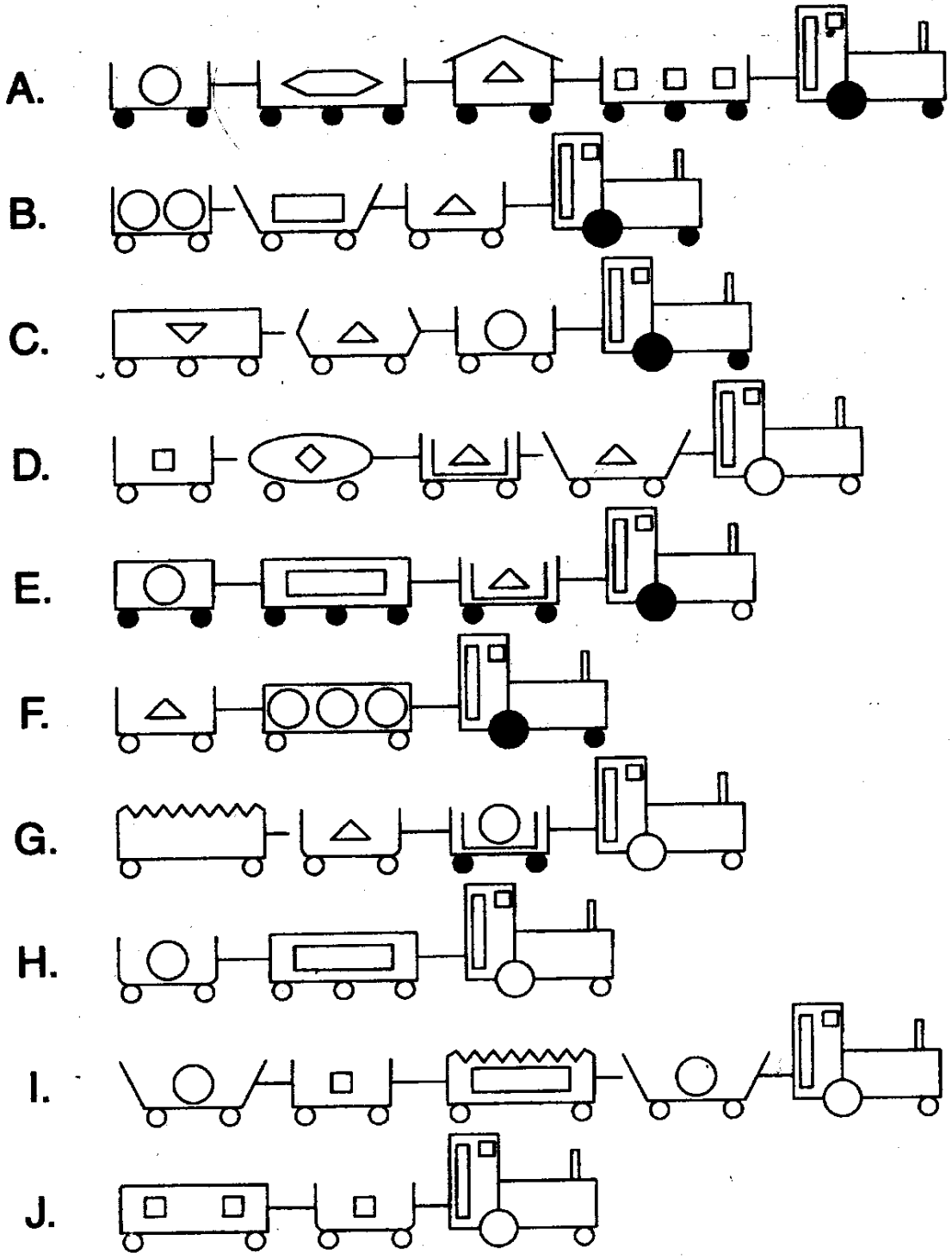
1. Suchen Sie, ausgehend von obigem Dialog, die benötigten Klassen.
   Hinweis: Substantive sind ein Hinweis auf Klassen. Zum Teil bezeichnen Sie aber auch Objekte, dann muss die dazugehörige Klasse gefunden werden.
2. Suchen Sie die Assoziationen, welche die gefundenen Klassen verbinden, und benennen Sie diese.
   Hinweis: Welche Klasse muss welche andere Klasse kennen? Denken Sie an Beziehungen wie: Klasse A ist Teil von Klasse B, Klasse A muss mit Klasse B kommunizieren, etc.
3. Suchen Sie die Kardinalitäten und evtl. die Rollennamen.
4. Suchen Sie für die Klassen die Attribute.
   Hinweis: Attribute sind meist einfache Datentypen (Zahl, Text o.ä.). Sind sie komplizierter, ist eine Assoziation zu einer (neuen) Klasse besser.

B.5.1 Mögliche Lösung zum Arbeitsblatt 5: Erste Problemanalyse (OOA)



B.6 Arbeitsblatt 6: Problem der Klassifizierung

Aufgabe: Suchen Sie für das Problem der unten stehenden Graphik passende Klassen.

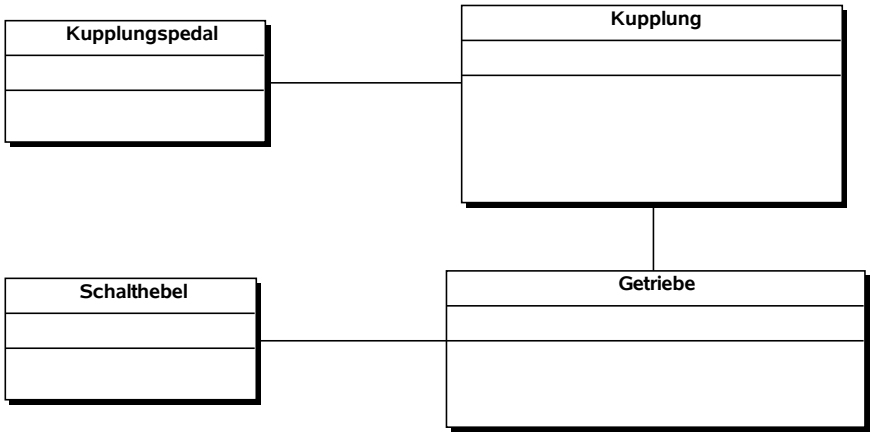


(nach [Bc94])

B.7 Arbeitsblatt 7: Finden von Methoden

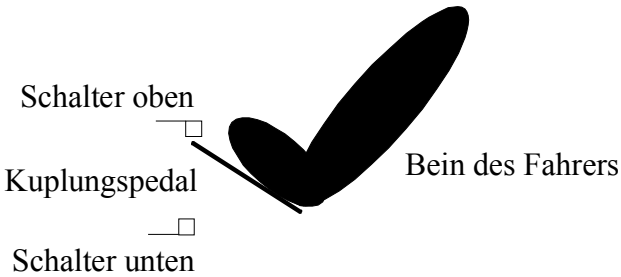
Ermitteln Sie die Methoden, welche beim Autofahren beim Schalten von einem Gang in den nächsten nötig sind. Überlegen Sie sich dabei genau die Abfolge der Tätigkeiten des Autofahrers. Gegeben ist unten stehendes Klassendiagramm.

Kupplungspedal:	Dieses wird vom Akteur *Fahrer* direkt betätigt.
Kupplung:	Hier erfolgt die Trennung des Motors vom Getriebe.
Schalthebel:	Mit diesem wählt der Akteur *Fahrer* den gewünschten Gang.
Getriebe:	Hier wird der gewünschte Zahnkranz ausgewählt.



Hinweis: Das Getriebe soll nachprüfen, ob gekuppelt wurde!

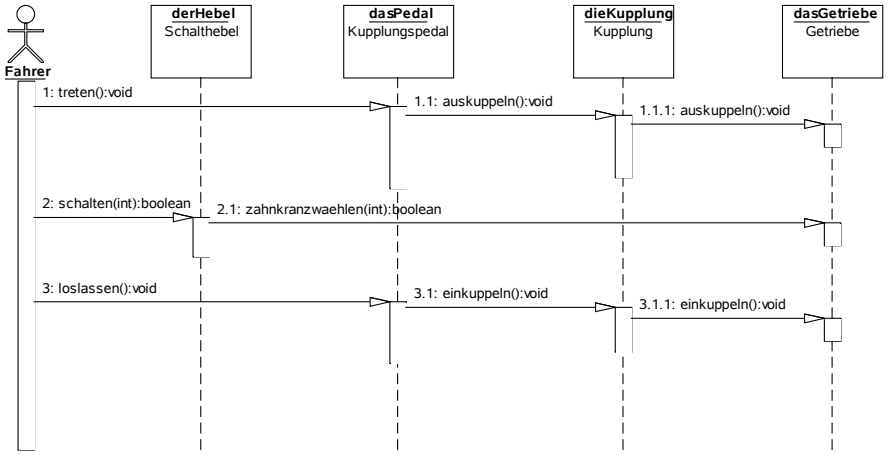
Aufbau Kupplungspedal:



B.7.1 Mögliche Lösungen zu Arbeitsblatt 7: Finden von Methoden

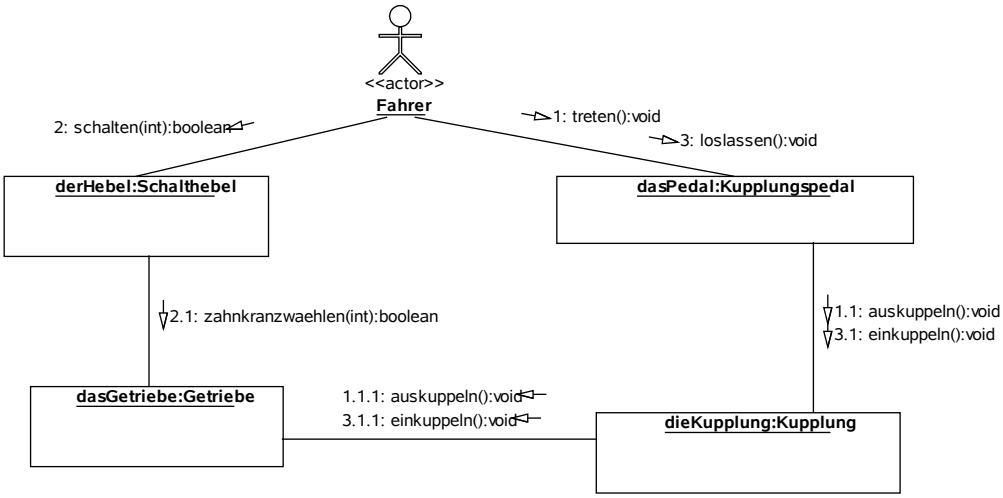
Schlechtere Lösung

Sequenzdiagramm:

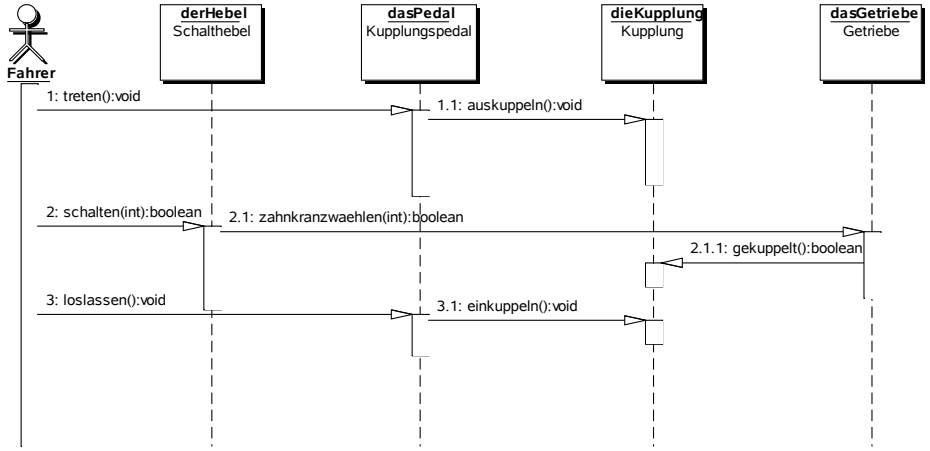


Der erste Designgrundsatz ist verletzt: Der Informationsexperte für das Kuppeln ist die Kupplung, deshalb muss vor dem Schalten bei der Kupplung nachgefragt werden. Auch der 3. Grundsatz ist verletzt. Das Getriebe hat nicht die Verantwortlichkeit, um sich den Zustand der Kupplung zu merken.

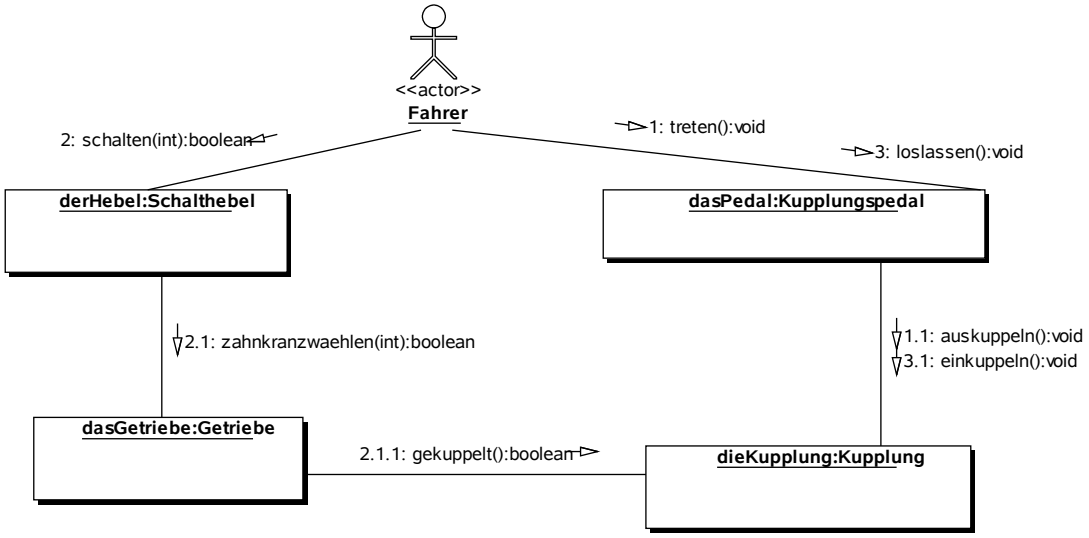
Kollaborationsdiagramm:



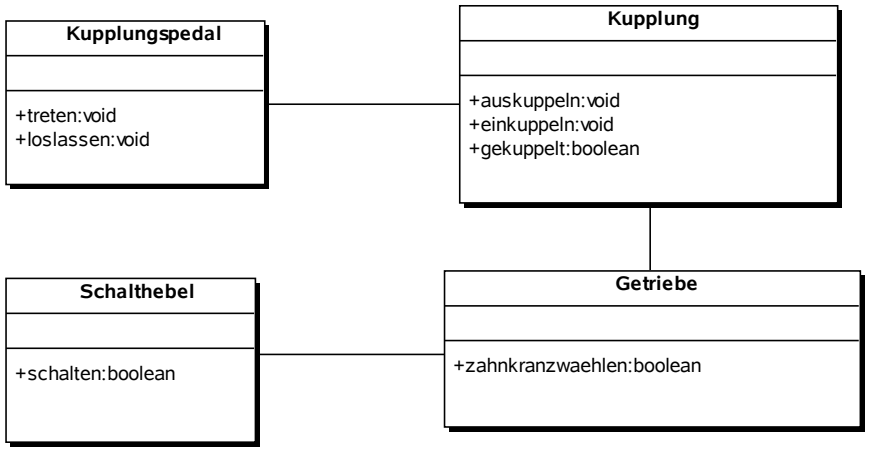
Bessere Lösung
Sequenzdiagramm:



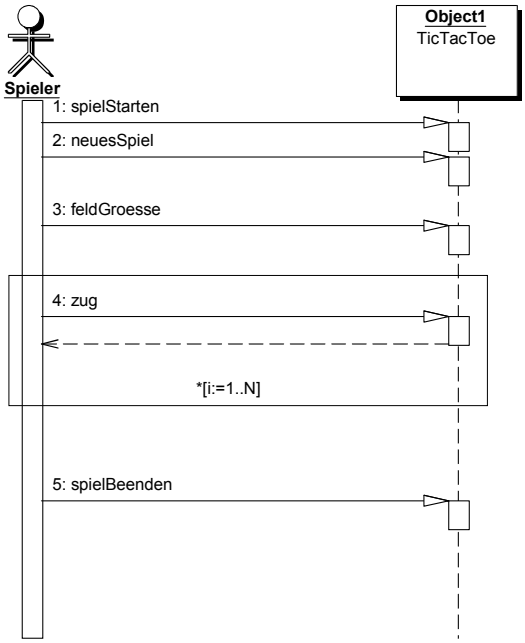
Kollaborationsdiagramm:



Klassendiagramm:



B.8 Arbeitsblatt 8: Design und Implementierung von TicTac-Toe

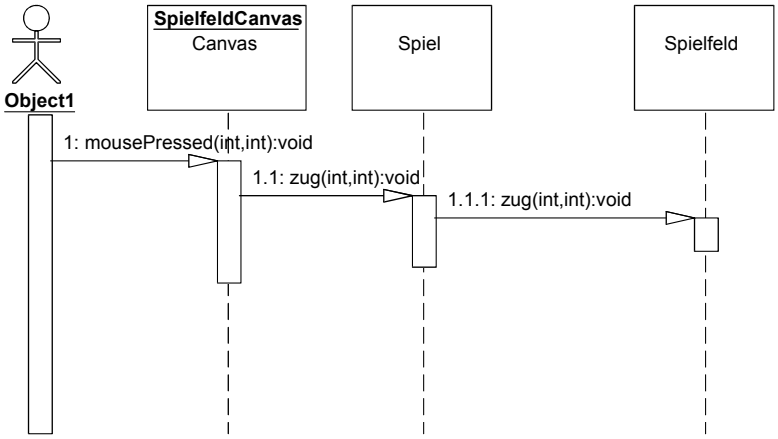


Ausgehend von der Analyse des TicTacToe-Spiels soll nun das Design entworfen werden. Beim Design steht nun die Frage des WIE im Vordergrund.
Zuerst werden alle Ereignisse, die von außerhalb des Systems (d.h. außerhalb unseres Programms) kommen, noch einmal aufgelistet, z.B. mit einem Sequenzdiagramm (siehe rechts).

Für jedes größere dieser Ereignisse fertigt man nun ein Sequenzdiagramm an, z.B. die Botschaft *zug*:

1. Ausgelöst wird dieses Ereignis durch das Anklicken eines Feldes auf unserem Spielfeld.
2. Welche Softwareklasse soll dieses Ereignis entgegennehmen? → eine Klasse, die das Fenster unseres Spielfeldes repräsentiert (Klassenname: *SpielfeldCanvas*). Die Klasse *SpielfeldCanvas* (Canvas = Leinwand) soll aber nur für die Anzeige und die erste Entgegennahme der Ereignisse zuständig sein.
3. Wer erhält dann als Nächstes eine Botschaft? Die Klasse *Spiel* soll i.A. die Vermittlung zwischen der Anzeige und der Verarbeitung übernehmen.
4. Wer ist innerhalb der Verarbeitung zuständig dafür, dass ein neuer Spielstein gesetzt wird? → die Klasse *Spielfeld*, denn dort soll alles zusammenlaufen, was sich auf das Spielfeld bezieht.

Wir haben also bisher folgende Klassen gefunden: *Canvas*, *Spiel*, *Spielfeld*. Das Sequenzdiagramm sieht bisher so aus:



1. Ergänzen Sie nun das Sequenzdiagramm, um die Sequenz zur Durchführung eines Zugs abzuschließen. Vergessen Sie nicht, dass es eine Umrechnung von Bildschirmkoordinaten zu Koordinaten des Spiels geben muss und dass nach Beendigung der Sequenz das

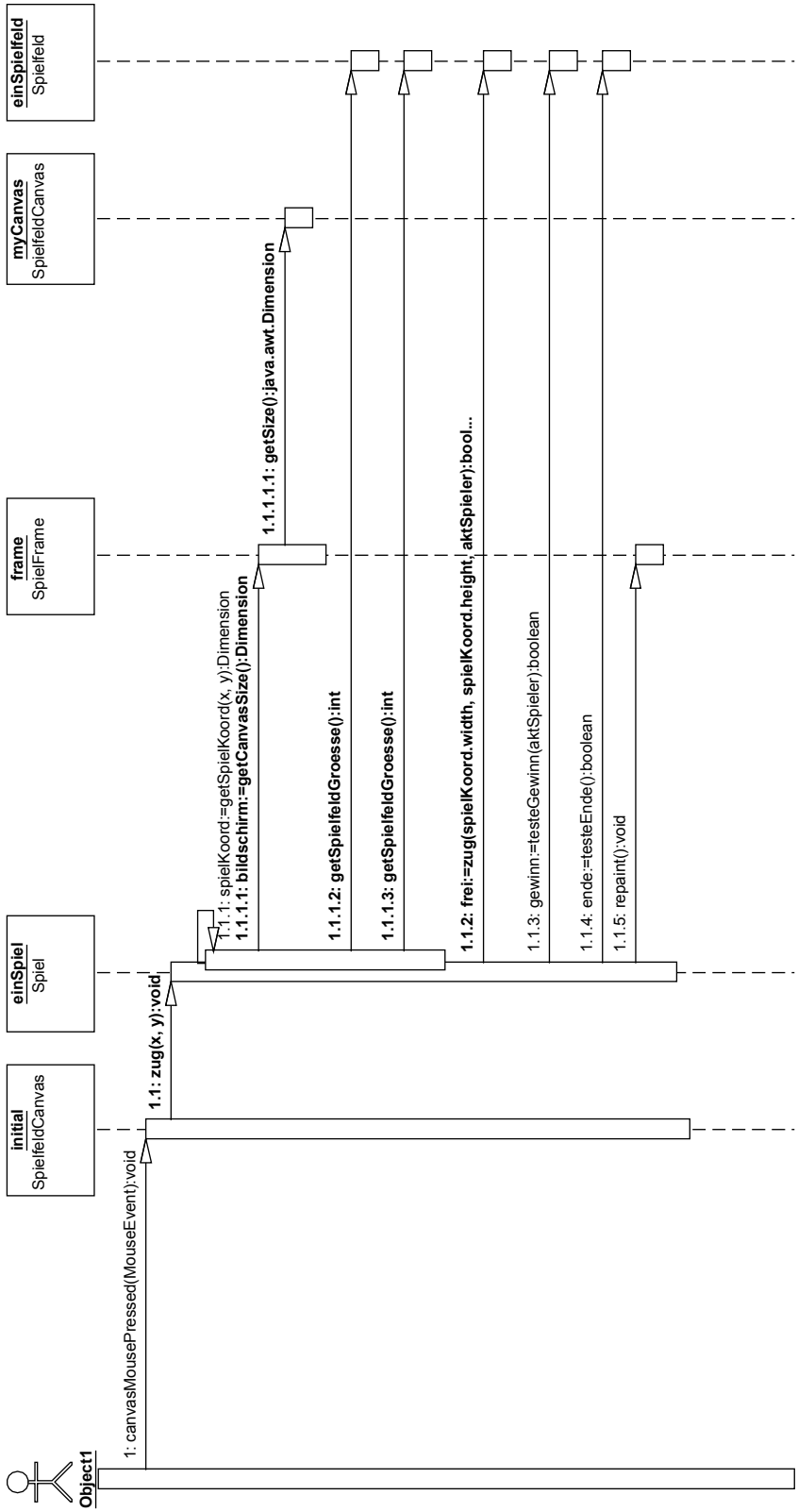
Bild neu gezeichnet werden muss. Überlegen Sie jedes Mal, wer der Informationsexperte ist, um die Botschaft an das richtige Ziel zu versenden!

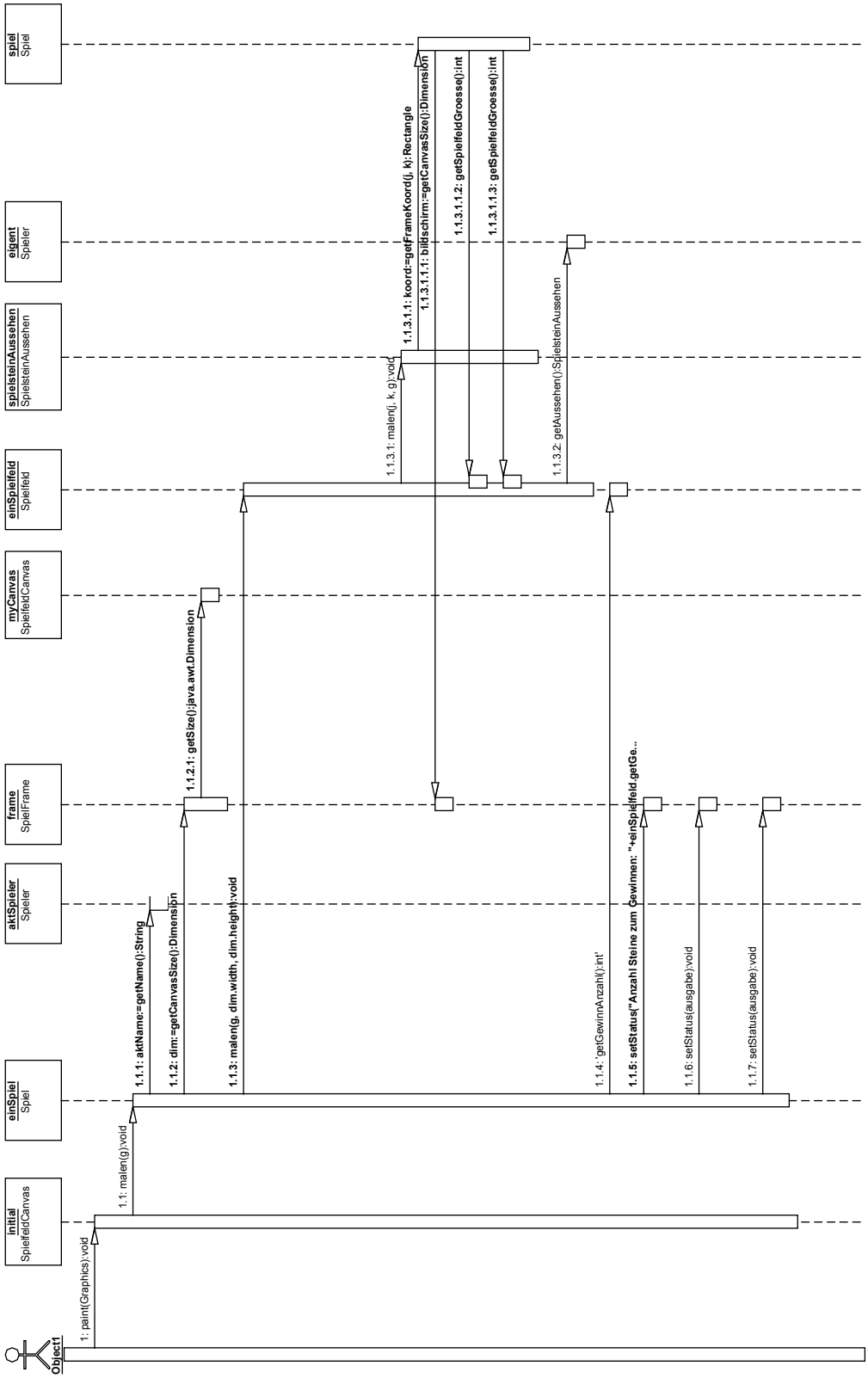
2. Zeichnen Sie nun das passende Klassendiagramm. Wenn eine Methode eine Methode eines anderen Objekts verwendet, muss sie dieses Objekt kennen. Eine Möglichkeit ist eine Assoziation. Zeichnen Sie auch diese ein und benennen Sie diese. Eine andere Möglichkeit ist, sie über einen weiteren Methodenaufruf von einem anderen Objekt zu erhalten.
3. Arbeiten Sie nun an der Sequenz für das Ereignis *paint()* des Akteurs System, welches unser Programm auffordert, sich neu zu zeichnen. Hier benötigen wir unter anderem die Umrechnung von Spielkoordinaten zu Bildschirmkoordinaten.
4. Ergänzen Sie nun das Klassendiagramm.
5. Wie soll unser Programm gestartet werden? Welche Klasse soll welches Objekt erzeugen? Es wird festgelegt, welche Klasse welches Objekt beim Starten erzeugen soll. Dieses Diagramm sollte immer das Letzte sein, da erst zum Schluss bekannt ist, welche Objekte mit welchen Werten initialisiert werden sollen.
6. Ergänzen Sie das Klassendiagramm.
7. Für die Implementierung sollten Sie folgende Reihenfolge einhalten:
   a. Zuerst werden die Klassen implementiert, welche keine bzw. am wenigsten andere Klassen zur Arbeit benötigen.
   b. Dann kommen die Klassen, welche mehr fremde Klassen benötigen.
   c. Zuletzt kommen die Klassen, welche am meisten andere Klassen benötigen.

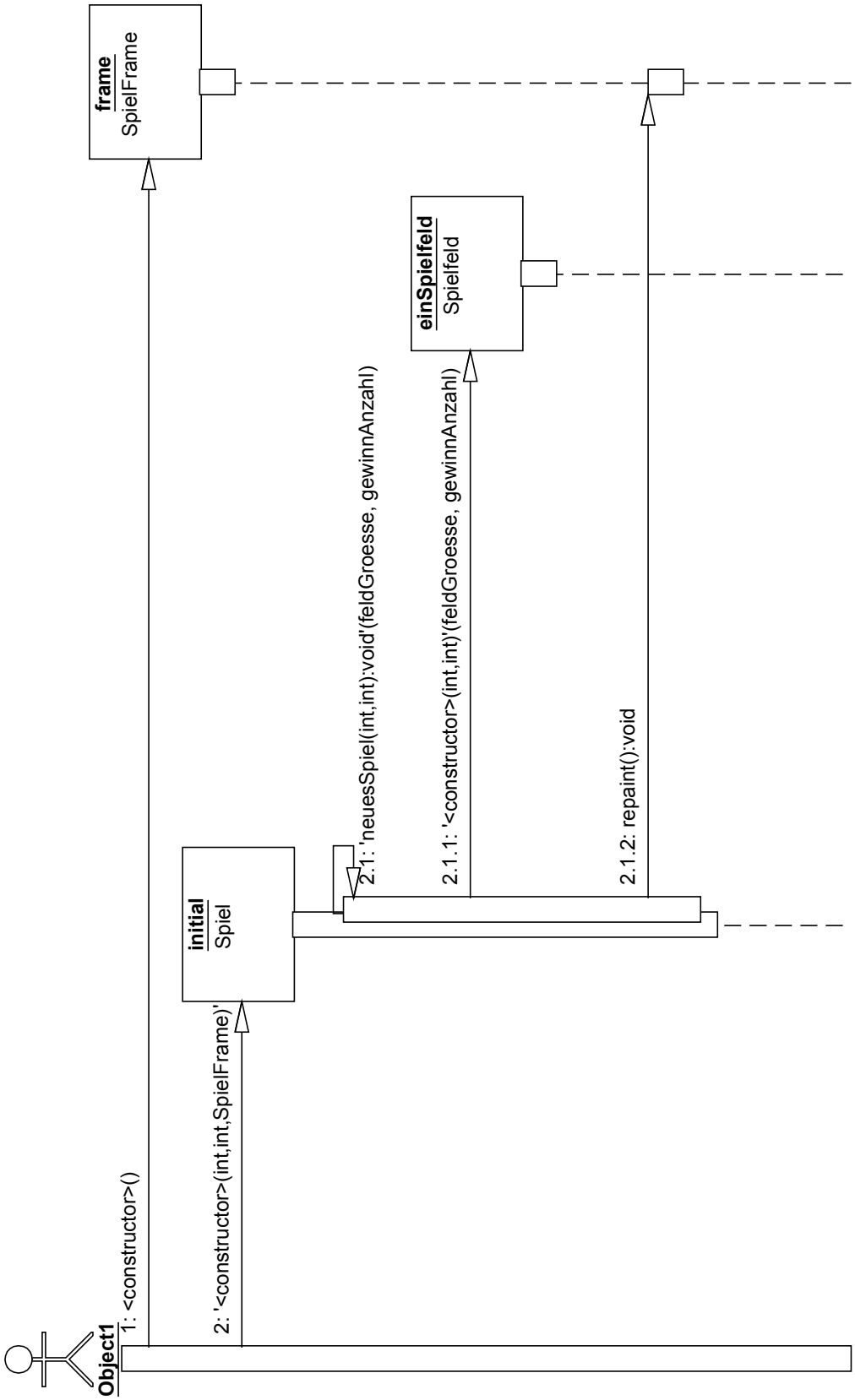
   Dies ist wichtig zum Testen der Klassen!
8. Bewerten Sie nun, wie schwer es ist, die einzelnen Methoden zu programmieren. Teilen Sie hiervon und vom vorhergehenden Punkt ausgehend den einzelnen Gruppenmitgliedern Klassen zu, die implementiert werden sollen, sodass die Arbeit gerecht verteilt ist.
9. Jeder ist nun für seine Klasse verantwortlich. Zuerst sollte aber nicht die Klasse sondern ein kleines Testprogramm für die Klasse programmiert werden. Also ein kleines Programm, welches die Klasse testet und auf die richtige Funktion überprüft.

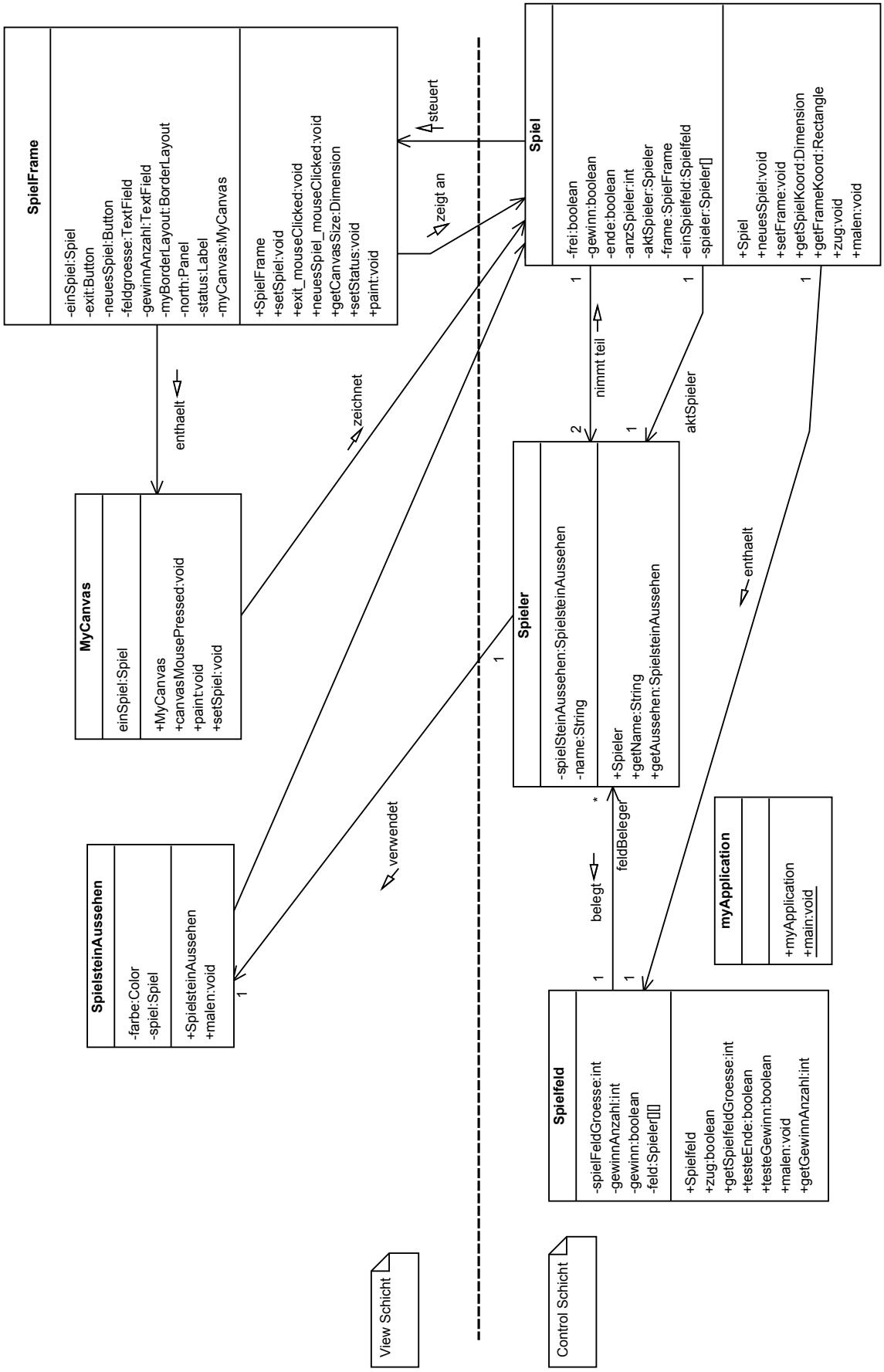
   Sollten Sie feststellen, dass Sie weitere Informationen aus einer anderen Klasse benötigen, müssen Sie mit dem entsprechenden Gruppenmitglied Kontakt aufnehmen! Alle Änderungen müssen im Klassendiagramm dokumentiert werden!
10. Bauen Sie dann die einzelnen Klassen Stück für Stück zum gemeinsamen Programm zusammen. Begnügen Sie sich vorerst mit der zuvor geplanten einfachen Funktionalität.
11. Wenn Ihr Programm funktioniert, können Sie Ihr Produkt verfeinern. Beginnen Sie hierfür wieder bei der Analyse: Was möchte ich verfeinern? Arbeiten Sie dann die Änderungen in das Design ein und zum **Schluss** in den Code!

B.8.1 Mögliche Lösung zu Arbeitsblatt 8: Design und Implementierung von TicTacToe




Object1
initial SpielfeldCanvas
einSpiel Spiel
aktSpieler Spieler
frame SpielFrame
myCanvas SpielfeldCanvas
einSpielfeld Spielfeld
spielsteinAussehen SpielsteinAussehen
eigent Spieler
spiel Spiel
1: paint(Graphics):void
1.1: malen(g):void
1.1.1: aktName:=getName():String
1.1.2: dim:=getCanvasSize():Dimension
1.1.2.1: getSize():java.awt.Dimension
1.1.3: malen(g, dim.width, dim.height):void
1.1.3.1: malen(j, k, g):void
1.1.3.1.1: koord:=getFrameKoord(j, k):Rectangle
1.1.3.1.1.1: bildschirm:=getCanvasSize():Dimension
1.1.3.1.1.2: getSpielfeldGroesse():int
1.1.3.1.1.3: getSpielfeldGroesse():int
1.1.3.2: getAussehen():SpielsteinAussehen
1.1.4: "getGewinnAnzahl():int"
1.1.5: setStatus("Anzahl Steine zum Gewinnen: "+einSpielfeld.getGe...
1.1.6: setStatus(ausgabe):void
1.1.7: setStatus(ausgabe):void


Object1
1: <constructor>()
2: '<constructor>(int,int,SpielFrame)'
frame
SpielFrame
initial
Spiel
2.1: 'neuesSpiel(int,int):void'(feldGroesse, gewinnAnzahl)
einSpielfeld
Spielfeld
2.1.1: '<constructor>(int,int)'(feldGroesse, gewinnAnzahl)
2.1.2: repaint():void

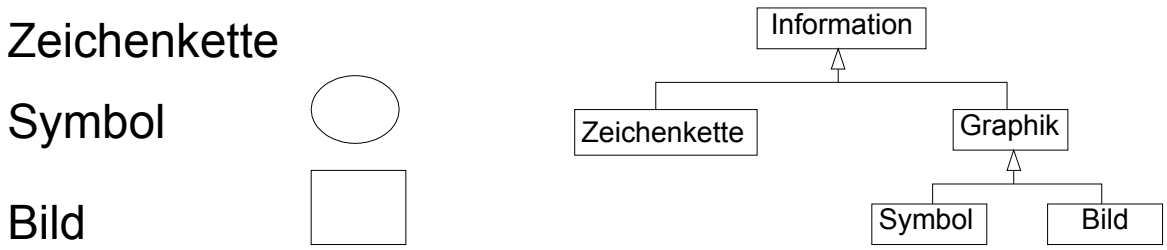

SpielFrame
-einSpiel:Spiel
-exit:Button
-neuesSpiel:Button
-feldgroesse:TextField
-gewinnAnzahl:TextField
-myBorderLayout:BorderLayout
-north:Panel
-status:Label
-myCanvas:MyCanvas
+SpielFrame
+setSpiel:void
+exit_mouseClicked:void
+neuesSpiel_mouseClicked:void
+getCanvasSize:Dimension
+setStatus:void
+paint:void
steuert
zeigt an
Spiel
-frei:boolean
-gewinn:boolean
-ende:boolean
-anzSpieler:int
-aktSpieler:Spieler
-frame:SpielFrame
-einSpielfeld:Spielfeld
-spieler:Spieler[]
+Spiel
+neuesSpiel:void
+setFrame:void
+getSpielKoord:Dimension
+getFrameKoord:Rectangle
+zug:void
+malen:void
enthaelt
zeichnet
nimmt teil
aktSpieler
MyCanvas
einSpiel:Spiel
+MyCanvas
+canvasMousePressed:void
+paint:void
+setSpiel:void
Spieler
-spielSteinAussehen:SpielsteinAussehen
-name:String
+Spieler
+getName:String
+getAussehen:SpielsteinAussehen
enthaelt
verwendet
SpielsteinAussehen
-farbe:Color
-spiel:Spiel
+SpielsteinAussehen
+malen:void
belegt
feldBeleger
myApplication
+myApplication
+main:void
Spielfeld
-spielFeldGroesse:int
-gewinnAnzahl:int
-gewinn:boolean
-feld:Spieler[][]
+Spielfeld
+zug:boolean
+getSpielfeldGroesse:int
+testeEnde:boolean
+testeGewinn:boolean
+malen:void
+getGewinnAnzahl:int
View Schicht
Control Schicht

B.9 Arbeitsblatt 9: Klassifizieren und Generalisieren

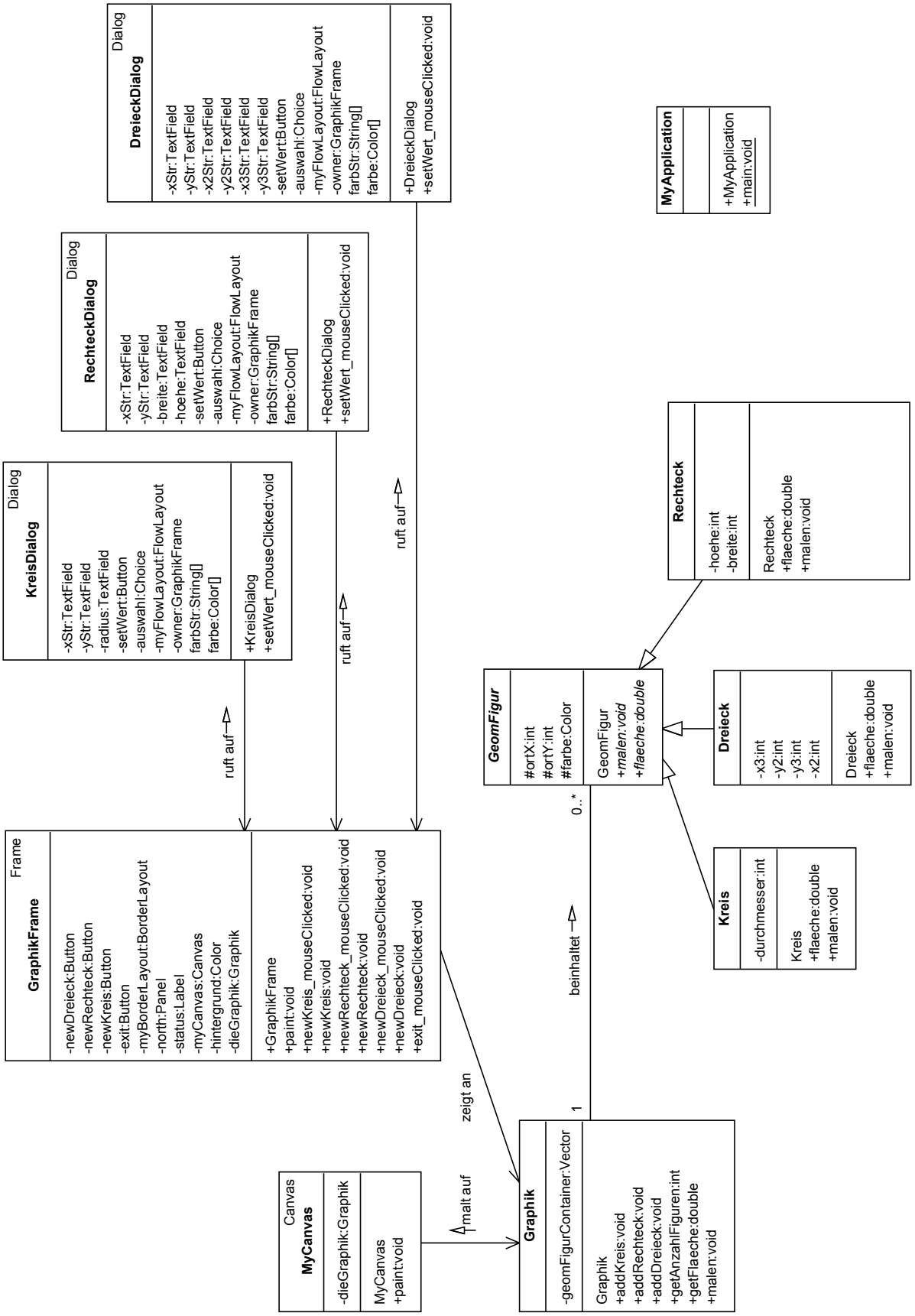
Aufgabe:
1. Suchen Sie die Objekte.
2. Suchen Sie nach Klassen.
3. Suchen Sie nach Gemeinsamkeiten zwischen den Klassen und generalisieren Sie!
4. Stellen Sie das Klassendiagramm dar.

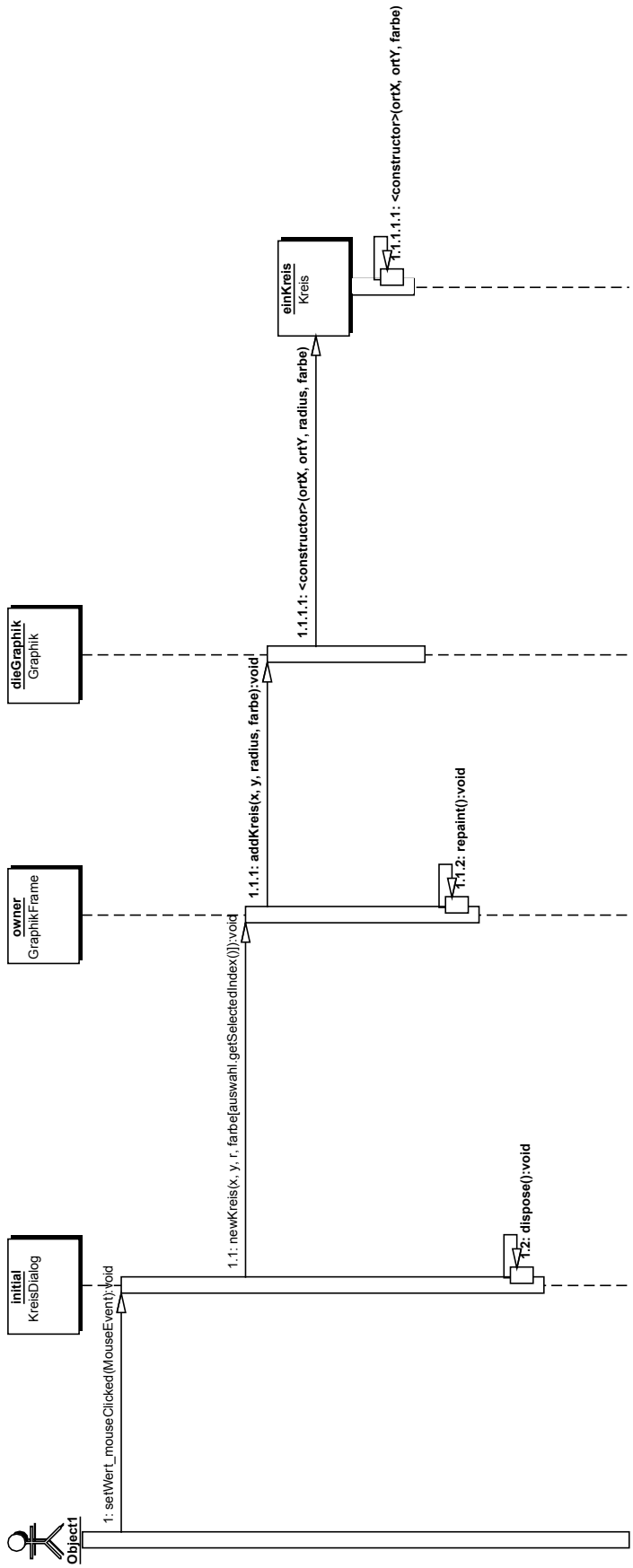
(nach [Wi])

B.9.1 Mögliche Lösung zu Arbeitsblatt 9: Klassifizieren und Generalisieren



B.10 Mögliche Lösung zum Graphik-Projekt




Object1
initial
KreisDialog
owner
GraphikFrame
dieGraphik
Graphik
einKreis
Kreis
1: setWert_mouseClicked(MouseEvent):void
1.1: newKreis(x, y, r, farbe[auswahl.getSelectedIndex()]):void
1.1.1: addKreis(x, y, radius, farbe):void
1.1.1.1: <constructor>(ortX, ortY, radius, farbe)
1.1.1.1.1: <constructor>(ortX, ortY, farbe)
1.1.2: repaint():void
1.2: dispose():void

B.11 Arbeitsblatt 10: Ein dynamisches Array (für Java)

Speichern der Objekte

In Java (und auch in C++) gibt es eine Klasse *Vector*, mit der dynamisch Daten abgelegt werden können. Diese Klasse kann Objekte der Klasse *Object* abspeichern, d.h. Objekte welche in *Vector* abgelegt werden sollen, müssen von der Klasse *Object* abgeleitet (vererbt) werden.
Die Klasse *Vector* befindet sich im Paket *java.util*.
Mit der Methode *add* können neue Elemente abgelegt werden.
Beispiel:

```
import java.util.* ;
public class Dummy extends Object {
...
}
...
Vector einVektor = new Vector();
Dummy einDummy = new Dummy();
einVektor.add(einDummy);
//oder direkt
einVektor.add(new Dummy());
```

Die Klasse *Vector* bietet weitere Methoden, um z.B. das Element an der n. Stelle zu holen, zu löschen oder einzufügen, die Anzahl der gespeicherten Objekte zu ermitteln oder ein Objekt zu suchen etc. (siehe Hilfe).

Alle Objekte abrufen

Möchte man auf die Gesamtheit aller abgespeicherten Objekte zugreifen, benötigt man eine Iteratorklasse (Iteration = Wiederholung), hier *Enumeration*.
Mit der Methode *elements* der Klasse *Vector* erhält man diesen Iterator für das Speicherobjekt.
Mit der Methode *nextElement* der Klasse *Enumeration* kann man sich Stück für Stück alle gespeicherten Objekte holen und mit der Methode *hasMoreElements* kann man das Ende abfragen.

Beispiel (die Objekte wurden mit obigem Beispiel abgelegt):

```
Enumeration enum = einVektor.elements();
while (enum.hasMoreElements()) {
     Dummy einDummy = (Dummy)enum.nextElement();
     //mit dem Objekt einDummy kann nun gearbeitet werden
     }
```

Der Typecast in der 3. Zeile des Beispiels ist wichtig. Ohne ihn wird nur ein Objekt der Klasse *Object* zurückgegeben und damit sind dann alle Methoden und Eigenschaften der Klasse *Dummy* nicht vorhanden!

Aufgabe:

Ergänzen Sie das Programm *Graphik*, so dass eine unbegrenzte Anzahl von Figuren abgespeichert werden kann (nur begrenzt durch die Größe des Arbeitsspeichers).

B.12 Arbeitsblatt 11: Zustandsautomaten

Aufgabe 1: Alarmanlage
Eine Alarmanlage für ein Haus soll entworfen werden. Sie hat folgende Zustände:

- aus,
- scharf,
- Alarm.

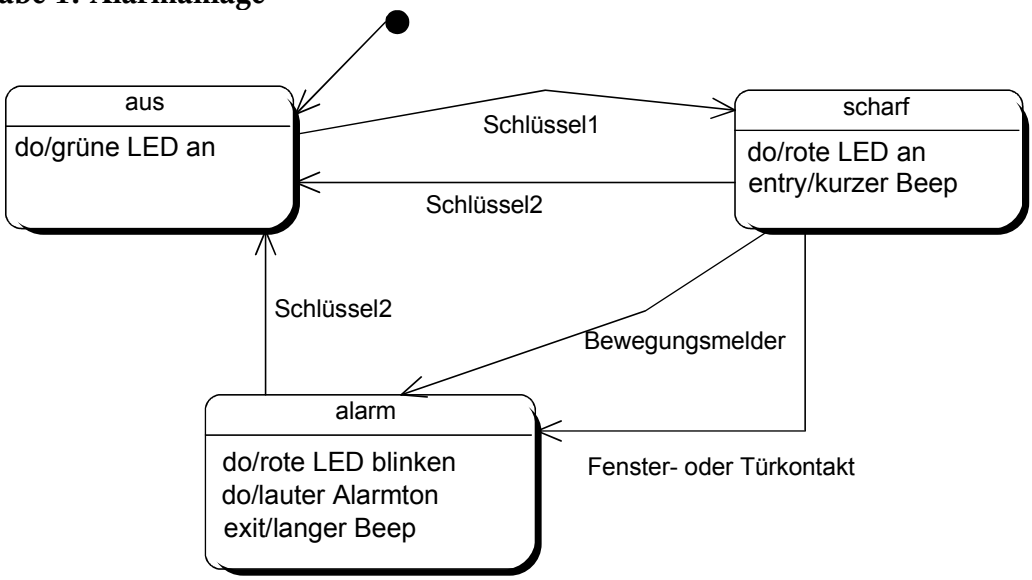
Die Übergänge zwischen den Zuständen erfolgt durch einen Schlüsselschalter (*Schlüssel1*, scharf stellen) und Alarmkontakte an Türen und Fenstern bzw. Bewegungsmeldern.
Ausschalten kann man die Alarmanlage nur mit einem weiteren Schlüssel (*Schlüssel2*).
Auch der Alarmzustand kann nur mit *Schlüssel2* verlassen werden.
Entwerfen Sie das Zustandsdiagramm, damit die Alarmanlage korrekt arbeitet!

Aufgabe 2: Paritätserkennung
Bei der Übertragung von Daten von einem Gerät zu einem anderen Gerät kann es immer wieder vorkommen, dass sich durch äußere Störungen einzelne Bits verändern.
Ein Verfahren, um dies zu erkennen, ist die Paritäts-Erkennung. Dabei wird gezählt, ob die Anzahl der *1*en in einem Datenwort gerade oder ungerade ist.
Es soll ein Automat entwickelt werden, der für eine unendlich lange Bit-Folge jeweils anzeigt, ob die bisher gesendeten Daten eine gerade oder eine ungerade Parität aufweisen.
Am Anfang ist die Parität gerade.

1. Entwerfen Sie das Zustandsdiagramm.
2. Geben Sie die Übergangs-Tabelle δ und die Ausgangstabelle λ an.

B.12.1 Mögliche Lösung zum Arbeitsblatt 11: Zustandsdiagramme

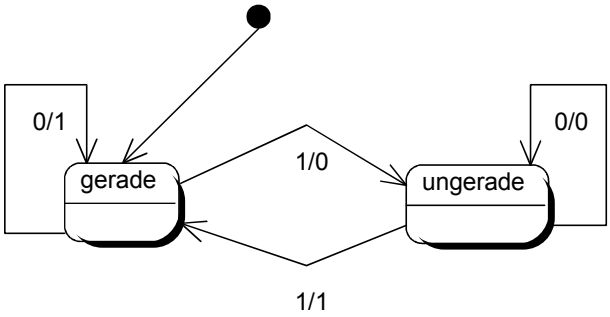
Aufgabe 1: Alarmanlage



Aufgabe 2: Paritätserkennung

Ausgangstabelle λ:

		Ereignis 0	Ereignis 1
Zustand	gerade	1	0
	ungerade	0	1



Übergangstabelle δ:

		Ereignis 0	Ereignis 1
Zustand	gerade	→ gerade	→ ungerade
	ungerade	→ ungerade	→ gerade

B.13 Arbeitsblatt 12: Maier-Suchmaschine

Eine Maschine, welche alle möglichen Variationen des Namens Maier findet, soll entwickelt werden. In Programmiersprachen wie Perl [El99] oder Python kann dies mit regulären Ausdrücken erledigt werden. Auch manches Betriebssystem wie z.B. Linux bietet Shell-Befehle, welche mit regulären Ausdrücken die Suche vereinfachen.

- Mögliche Schreibweisen: maier, mayer, meier, meyer
- Regulärer Ausdruck: m[ae][iy]er

Die Angaben in der eckigen Klammer bedeuten, dass die Buchstaben alternativ vorkommen können. Zur Vereinfachung sollen die Zeichen zuvor in Kleinbuchstaben umgewandelt werden.
Die Eingabe des zu durchsuchenden Textes soll zunächst über ein Editor-Feld erfolgen. Nach Betätigung eines Buttons soll nach allen vorkommenden Formen gesucht und in einem Ausgabefeld die Anzahl der gefundenen Stellen angezeigt werden.

Hinweis: Reguläre Ausdrücke bieten noch viele weitere Möglichkeiten.

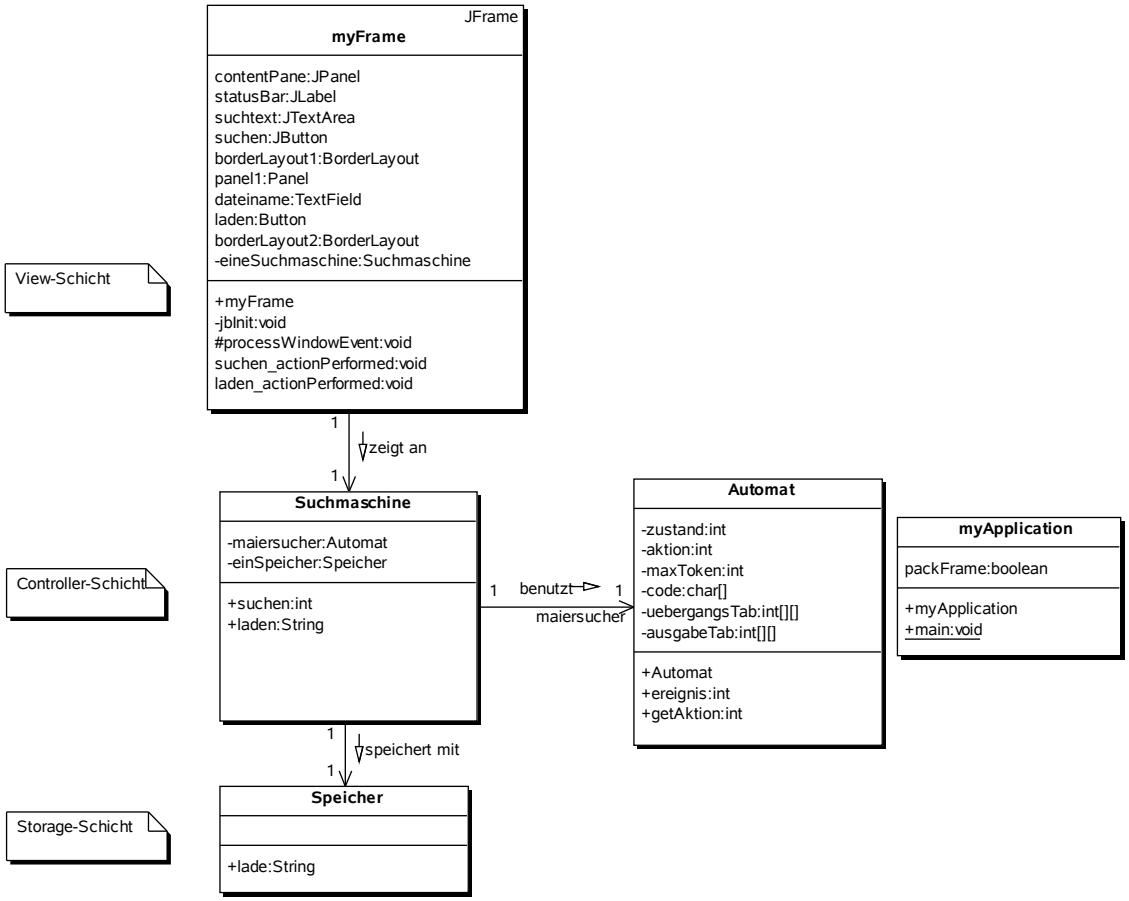
Aufgaben:

1. Entwerfen Sie das Klassendiagramm.
2. Entwerfen Sie das Zustandsdiagramm.
3. Entwickeln sie aus dem Zustandsdiagramm die Übergangs- und Ausgangstabelle.
4. Schreiben Sie das Programm.

5. Erweitern sie das Klassendiagramm, so dass auch ein Text in einer Datei gelesen und untersucht werden kann.
6. Erweitern Sie entsprechend Ihr Programm.

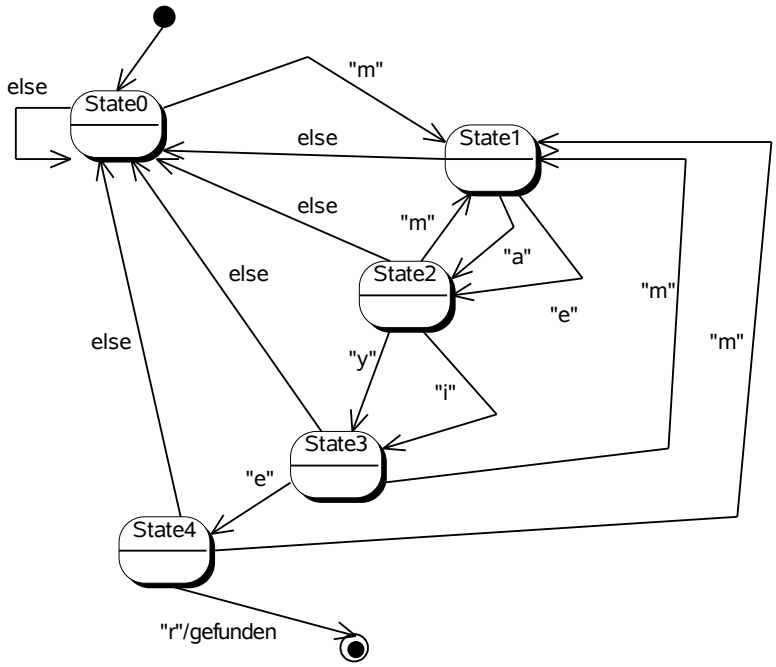
B.13.1 Mögliche Lösung zum Arbeitsblatt 12: Maier-Suchmaschine

Klassendiagramm:



Hinweis:
Unter Verwendung des CVS-Patterns ergibt sich die Verwendung der Control-Klasse *Suchmaschine*, da die View-Klasse *myFrame* nicht direkt auf die Storage-Schicht zugreifen sollte (Vorgabe des CVS-Patterns). Deswegen muss die Klasse *Suchmaschine* eingeführt werden. Die Klasse *Automat* ist dafür eine schlechte Wahl, da die Designrichtlinien *Informationsexperte* und *Hoher Zusammenhalt* sonst verletzt würden.

Zustandsdiagramm:



Anstatt bei einem Token *r* vom Zustand 4 in den Endzustand (Zustand 5) zu gelangen, ist es auch möglich, in den Zustand 0 zu gehen. Dann wird bei diesem Ereignis die Zustandsmaschine automatisch neu gestartet.
Die beiden unten stehenden Tabellen gehen hiervon aus.

Ausgangstabelle λ:

		Token						
		m	*a*	*e*	*i*	*y*	*r*	else
		0	1	2	3	4	5	6
Zustand	0	n	n	n	n	n	n	n
	1	n	n	n	n	n	n	n
	2	n	n	n	n	n	n	n
	3	n	n	n	n	n	n	n
	4	n	n	n	n	n	g	n
	5=0							

n (bzw. 0) = nicht gefunden; g (bzw. 1) = gefunden

Übergangstabelle δ:

		Token						
		m	*a*	*e*	*i*	*y*	*r*	*else*
		0	1	2	3	4	5	6
Zustand	0	1	0	0	0	0	0	0
	1	1	2	2	0	0	0	0
	2	1	0	0	3	3	0	0
	3	1	0	4	0	0	0	0
	4	1	0	0	0	0	5	0
	5=0							

B.14 Arbeitsblatt 13: Anwendungsfälle für die Ligaverwaltung

Aufgabe:
Für einen Sportsender soll ein Programm zur Erstellung einer Tabelle zur Darstellung der Rangfolge in der Liga geschrieben werden.

1. Ermitteln Sie alle Anwendungsfälle!
2. Geben Sie für die Anwendungsfälle die Beschreibung an!

C Fragebögen

C.1 Fragebogen zur Erfassung des Schülerumfeldes

Liebe Eltern und Schüler,

ein Schwerpunktthema am neuen informationstechnischen Gymnasium ist die »Objektorientierte Analyse und Design«. Dieses noch sehr junge Thema bedarf noch einiger Untersuchungen zur Umsetzung. Deshalb habe ich mich entschlossen, zusammen mit der Universität Heidelberg in einer wissenschaftlichen Arbeit ein didaktisches Konzept hierfür zu erarbeiten. In dieses Konzept flossen meine mehrjährigen Erfahrungen in diesem Gebiet als Lehrer an der Werner-Siemens-Schule in Stuttgart ein, welche ich zum Teil bereits an der Staatlichen Akademie für Lehrerfortbildung in Esslingen an die Kollegen weitergegeben habe.
Um diesen Entwurf zu überprüfen, soll er unter anderem in der Klasse Ihrer Tochter/Ihres Sohns umgesetzt werden. Anhand von Fragebögen, Klassenarbeiten und Unterrichtsbesuchen werde ich dieses Konzept dann beurteilen. Dabei stehen nicht die Schüler, sondern der Entwurf im Vordergrund, da er seine Stärke unabhängig vom Lernenden und Lehrenden zeigen soll.
Die Ergebnisse der wissenschaftlichen Arbeit sollen veröffentlicht werden, um so allen Lehrern zugänglich zu sein. Die Untersuchungsergebnisse werden anonymisiert veröffentlicht, so dass keinerlei Rückschlüsse auf Einzelpersonen möglich sein werden. Die Einzeldaten werden spätestens ein Jahr nach der Veröffentlichung der Arbeit vernichtet.
Eine Genehmigung des Oberschulamts und des Schulleiters für diese Erhebung liegt vor.
Um die Untersuchung durchführen zu können, benötige ich jedoch auch Ihre Zustimmung. Deshalb bitte ich Sie, den unteren Abschnitt unterschrieben (bei Minderjährigen von den Erziehungsberechtigten) in den Unterricht von ..... mitzubringen.
Die erhobenen Daten werden nur im Rahmen dieser wissenschaftlichen Arbeit verwendet. Eine weiter gehende Nutzung ist ausgeschlossen. Eine Nichtteilnahme Ihrerseits führt zu keinerlei Nachteilen. Die Teilnahme ist freiwillig. Ich möchte sie jedoch noch einmal bitten, diese Genehmigung zu erteilen. Ohne diese von Ihnen unterschriebene Genehmigung bin ich nicht in der Lage, diese Arbeit durchzuführen. Auch ein sorgfältig entworfenes Konzept muss auf seine Richtigkeit überprüft werden.
Weiterhin bitte ich Sie, den beigelegten Fragebogen auszufüllen. Auch für diesen gilt, dass eine Nichtabgabe zu keinerlei Nachteilen für Ihre Tochter/Ihren Sohn führt. Aber auch hier möchte ich Sie bitten, den Fragebogen abzugeben, er hilft mir, evtl. auftretende Zusammenhänge mit dem Schülerumfeld in der Untersuchung auszuschließen. Weitere Fragen zum persönlichen und privaten Umfeld werde ich nicht erheben.
Ich möchte mich bereits im Vorfeld für Ihre Mitarbeit bedanken. Sollten Sie noch Fragen an mich haben, können Sie mich unter der E-Mail *erhebung@wss-stuttgart.de* erreichen.

Mit freundlichen Grüßen
StR Walter Kicherer

Ich bin damit einverstanden, dass meine Tochter/mein Sohn ..... an der Untersuchung zur »Objektorientierte Analyse und Design« von Herrn Kicherer teilnimmt.

....
Ort, Datum Unterschrift (bei Minderjährigen der Erziehungsberechtigten)

Fragebogen zur Erfassung des Schülerumfeldes

Name der Schülerin/des Schülers .....

Eltern/Erziehungsberechtigte

	Vater	Mutter
1.) Erlernter Beruf	.....	.....
2.) Ausgeübter Beruf		
Freier Beruf (Arzt, Anwalt, etc.) Leiter von Unternehmen	☐	☐
Beamter, Angestellter	☐	☐
Landwirtschaft	☐	☐
Selbständig Gewerbetreibender	☐	☐
Facharbeiter	☐	☐
Arbeiter	☐	☐
Hausfrau, Hausmann	☐	☐
Rentner, Pensionär	☐	☐
ohne Beruf	☐	☐

Schüler

3.) Ist zu Hause ein PC vorhanden, auf dem für die Schule geübt werden kann?

Ja ☐ Nein ☐

Wenn ja, welche Software ist installiert? (bitte ankreuzen):

Textverarbeitung	☐
Programmiersprache C++	☐
Programmiersprache Java	☐
sonstige Programmiersprache (bitte angeben, welche)	☐.....
Werkzeuge zum Programmieren (z.B. um UML-Diagramme oder Struktogramme zu zeichnen)	☐

4.) Freizeitbeschäftigung:

4.1) Ich beschäftige mich in meiner Freizeit mit dem PC.	Ja ☐	Nein ☐
4.2) Ich programmiere in meiner Freizeit auf dem PC.	Ja ☐	Nein ☐
4.3) Ich spiele in meiner Freizeit auf dem PC.	Ja ☐	Nein ☐

C.2 Fragebogen zur Erfassung des Lehrer- und Klassenumfeldes

Name des Lehrers: .....

1. Lehrervorkenntnisse

1.1 Wie alt sind Sie?

< 30	30 ... 40	40 ... 50	50 ... 60	> 60

1.2 Welche Fachrichtung haben Sie studiert? .....

1.3 Wie viele Jahre ist das her? .....

1.4 Seit wie vielen Jahren sind Sie Lehrer? .....

1.5 Welche Fächerkombination haben Sie (Informatik, Nachrichtentechnik, ...)? .

1.6 Wie viele Jahre unterrichten Sie bereits das Themengebiet der Programmierung?

.....

1.7 Wie viele Jahre unterrichten Sie das Themengebiet des objektorientierten Softwareentwurfs? .....

1.8 Woher haben Sie Ihre **fachlichen** Kenntnisse (z.B. Studium, Lehrerfortbildung, Literatur, Internet, vorhergehende Berufserfahrung, ... mehrere Angaben möglich, Schwerpunkt bitte unterstreichen)? Aus dem Bereich

1.8a der Programmierung (allgemein)

.....

.....

1.8b des objektorientierten Softwareentwurfs

.....

.....

1.9 Schätzen Sie bitte Ihre **fachliche** Kompetenz (0 = keine Kenntnisse, 10 = sehr gut) im Bereich der **Verfahren** zum

1.9a strukturierten Programmieren:

.....

.....

1.9b objektorientierten Softwareentwurf:

.....

.....

1.10 Schätzen Sie bitte Ihre **pädagogische** Erfahrung (0 = keine Kenntnisse, 10 = sehr gut) im Bereich

1.10a der strukturierten Programmierung:

.....

1.10b des objektorientierten Softwareentwurfs:

.....

2. Rahmenbedingungen der Klasse

2.1 Wie viele Schüler sind in der Klasse (männlich/weiblich)? .....

2.2a Wie viele Stunden unterrichten Sie die Klasse im Fach IT?.....

2.2b Wie viele davon in Klassenteilung?.....

2.2c Wie viele davon in einem PC-Raum? .....

2.3a Wie viele Stunden unterrichten Sie die Klasse im Fach CT?.....

2.3b Wie viele davon in Klassenteilung?.....

2.3c Wie viele davon in einem PC-Raum?.....

2.4 Gibt es in der Klasse Schüler, die Schwierigkeiten mit der deutschen Sprache haben? Wenn ja, wie viele? .....

2.5 Haben Sie in dieser Klasse bereits in Klasse 11 die LPE *Strukturierte Programmierung* unterrichtet (Ja/Nein)? .....

3. Ausstattung

3.1 Wie viele PCs stehen den Schülern im PC-Raum zur Verfügung? .....

3.2 Wie groß ist das Verhältnis Schüler/PC im Unterricht? .....

3.3 Welche Software nutzen Sie im Unterricht?.....

3.3a) verwendete Programmierumgebung (IDE; z.B. JBuilder 4.0, Visual Studio, ...)

.....

verwendete CASE-Tools

3.3b) UML (z.B. Together, Rational, ...) .....

3.3c) Struktogrammeditoren .....

3.3d) sonstiges .....

3.4 Welche Software nutzen Sie zur Unterrichtsvorbereitung?

3.4a) verwendete Programmierumgebung (IDE; z.B. JBuilder 4.0, Visual Studio, ...)

.....

verwendete CASE-Tools

3.4b) UML (z.B. Together, Rational, ...) .....

3.4c) Struktogrammeditoren .....

3.4d) Sonstiges .....

IDE Integrated Development Enviroment, Integrierte Entwicklungsumgebung
CASE Computer Aided Software Engineering
UML Unified Modeling Language

C.3 Lehrerfragebogen nach Vermittlung der Einheit (Umsetzungsgruppe)

Name:.....

Folgende Fragen beziehen sich auf Ihre Erfahrungen nach Durchführung der gesamte LPE 5 *Objektorientierte Analyse und Design*.

1. Schätzen Sie bitte Ihre **fachliche** Kompetenz im Bereich der **Verfahren** zum
   1a strukturierten Programmieren:

   keine Kenntnisse. sehr gut.

0	1	2	3	4	5	6	7	8	9	10

   1b objektorientierten Softwareentwurf:

   keine Kenntnisse. sehr gut.

0	1	2	3	4	5	6	7	8	9	10

2. Schätzen Sie bitte Ihre **pädagogische** Erfahrung im Bereich
   2a der strukturierten Programmierung:

   keine Kenntnisse. sehr gut.

0	1	2	3	4	5	6	7	8	9	10

   2b des objektorientierten Softwareentwurfs:

   keine Kenntnisse. sehr gut.

0	1	2	3	4	5	6	7	8	9	10

3. Der Unterricht hat ...

   ... mich gelangweilt ... mir Spaß gemacht.

0	1	2	3	4	5	6	7	8	9	10

4. Die Schüler waren im Unterricht ...

   ... unmotiviert ... motiviert.

0	1	2	3	4	5	6	7	8	9	10

5. Die Schüler sind für das Abitur ...

   ... schlecht vorbereitet. ... gut vorbereitet.

0	1	2	3	4	5	6	7	8	9	10

6. Die Schüler sind für das Studium ...

   ... schlecht gerüstet. ... gut gerüstet.

0	1	2	3	4	5	6	7	8	9	10

7. Unterrichtete Stundenzahl zur Lehrplaneinheit *Objektorientierte Analyse und Design*:
   7a im Fach Informationstechnik:..... 7b im Fach Computertechnik:.....
8. Ich werde das Konzept im Großen und Ganzen in meinem Unterricht in Zukunft weiterverwenden.
   - o Ja
   - o Nein
   - o Nur Teile. Welche: ..... (ggf. auf der Rückseite weiter ausführen)
9. Welchen Eindruck hatten Sie von diesem Konzept? Was würden Sie ändern?
   .....(ggf. auf der Rückseite weiter ausführen)

C.4 Lehrerfragebogen nach Vermittlung der Einheit (Kontrollgruppe)

Name:.....

Folgende Fragen beziehen sich auf Ihre Erfahrungen nach Durchführung der gesamte LPE 5 *Objektorientierte Analyse und Design*.

1. Schätzen Sie bitte Ihre **fachliche** Kompetenz im Bereich der **Verfahren** zum
   1a strukturierten Programmieren:

   keine Kenntnisse. sehr gut.

0	1	2	3	4	5	6	7	8	9	10

   1b objektorientierten Softwareentwurf:

   keine Kenntnisse. sehr gut.

0	1	2	3	4	5	6	7	8	9	10

2. Schätzen Sie bitte Ihre **pädagogische** Erfahrung im Bereich
   2a der strukturierten Programmierung:

   keine Kenntnisse. sehr gut.

0	1	2	3	4	5	6	7	8	9	10

   2b des objektorientierten Softwareentwurfs:

   keine Kenntnisse. sehr gut.

0	1	2	3	4	5	6	7	8	9	10

3. Der Unterricht hat ...

   ... mich gelangweilt ... mir Spaß gemacht.

0	1	2	3	4	5	6	7	8	9	10

4. Die Schüler waren im Unterricht ...

   ... unmotiviert ... motiviert.

0	1	2	3	4	5	6	7	8	9	10

5. Die Schüler sind für das Abitur ...

   ... schlecht vorbereitet. ... gut vorbereitet.

0	1	2	3	4	5	6	7	8	9	10

6. Die Schüler sind für das Studium ...

   ... schlecht gerüstet. ... gut gerüstet.

0	1	2	3	4	5	6	7	8	9	10

7. Unterrichtete Stundenzahl zur Lehrplaneinheit *Objektorientierte Analyse und Design*:
   7a im Fach Informationstechnik:..... 7b im Fach Computertechnik:.....

C.5 Schülerfragebogen nach Vermittlung der Einheit

1. Ein(en) Beruf im Bereich des Softwareentwurfs...

... ist für mich undenkbar. ... kann ich mir vorstellen.

0	1	2	3	4	5	6	7	8	9	10

Die folgenden Fragen beziehen sich auf den Unterricht im Fach Informationstechnik *Objektorientierte Analyse und Design*.

2. Ich kann nach Ende der Unterrichtseinheit ...

... nicht besser programmieren. ... viel besser programmieren.

0	1	2	3	4	5	6	7	8	9	10

3. Das Thema fand ich ...

... langweilig. ... spannend.

0	1	2	3	4	5	6	7	8	9	10

4. Am **besten** verstanden habe ich (bitte maximal 2 Kreuze!) :

- o a) Klasse, Objekt, Zugriffsrechte, ...
- o b) Beziehungen, Rollen, Kardinalitäten, ...
- o c) Generalisierung, Vererbung, ...
- o d) Entwurf eines Klassenmodells, ...
- o e) Use Cases
- o f) UML (Unified Modeling Language)

5. Am s**chlechtesten** verstanden habe ich (bitte maximal 2 Kreuze!):

- o a) Klasse, Objekt, Zugriffsrechte, ...
- o b) Beziehungen, Rollen, Kardinalitäten, ...
- o c) Generalisierung, Vererbung, ...
- o d) Entwurf eines Klassenmodells, ...
- o e) Use Cases
- o f) UML (Unified Modeling Language)

6. Ich glaube, der Unterricht hat ...

... den **Lehrer** gelangweilt. ... dem **Lehrer** Spaß gemacht.

0	1	2	3	4	5	6	7	8	9	10

7. Der Unterricht hat ...

... **mich** gelangweit. ... **mir** Spaß gemacht.

0	1	2	3	4	5	6	7	8	9	10

C.6 Vor- und Nachtest
C.6.1 Aufgaben

Vortest: Frage 1 – 5.6
Nachtest: Frage 2 – 5.9
In Klammern ist jeweils der Bezug zu den Lernzielen angegeben.

Name: .....

In diesem Schulhalbjahr ist ein Schwerpunkt im Fach Informationstechnik der objektorientierte Softwareentwurf. Wie Sie bereits erfahren haben, soll in einer wissenschaftlichen Untersuchung ein neues Konzept erstellt werden. Um festzustellen, was Sie in diesem Halbjahr gelernt haben, um also zu überprüfen, ob das Konzept funktioniert, soll am Anfang dieser Test stehen. Am Ende des Schulhalbjahrs kann dann festgestellt werden, was Sie gelernt haben.
Viele Fragen werden Ihnen unbekannt vorkommen, bitte versuchen Sie jedoch auch diese Fragen nach bestem Wissen zu beantworten. Nur dann kann auch festgestellt werden, was Sie gelernt haben. **Die Ergebnisse dieses Tests werden nicht in Ihre Noten einfließen.**
Sie erfahren aber auch etwas über die Inhalte des Unterrichts im nächsten halben Jahr. Am Ende des Schulhalbjahres werden Sie sicherlich die meisten Fragen beantworten können.
Am Anfang des Tests steht noch eine Frage, welche sich auf das vorangegangene Schuljahr bezieht. Es soll damit festgestellt werden, ob Sie die Grundlage für das neue Thema besitzen.
Außer Aufgabe 1 können Sie die Lösungen direkt zu den Aufgaben notieren.
Bitte tragen Sie auf allen Blättern Ihren Namen ein. Kreuzen Sie danach an, wie gerne Sie programmieren.

	... ist völlig blöd										... ist echt geil
	0	1	2	3	4	5	6	7	8	9	10
Ich finde, Programmieren ...											

1. Strukturierter Softwareentwurf
   Zeichnen Sie auf einem extra Blatt das normgerechte Struktogramm für folgendes Problem:
   In einem Array mit 20 Elementen sind ganze Zahlen (Integer) gespeichert. Es soll die Summe über die im Array abgelegten **positiven** Zahlen gebildet werden. Ist also im Array eine negative Zahl (z.B. –5) abgelegt, so soll sie nicht in der Summe mit aufgenommen werden.

2. Objekt & Klasse

2.1 Zeichnen Sie eine Klasse *Auto* mit den Eigenschaften *sitzplaetze* und *aktGeschwindigkeit* und der Methode *beschleunigen*. (B, Y)

2.2 Ergänzen Sie die Klasse um eine weitere Eigenschaft und eine weitere Methode.(C)
2.3 Zeichnen Sie das Objektdiagramm für ein beliebiges Objekt der obigen Klasse. (C,Y)

2.4 Erklären Sie an einem beliebigen Beispiel den Unterschied in der objektorientierten Programmierung zwischen einer Klasse und einem Objekt. (B)

.....

.....

.....

2.5 Was versteht man in der Softwaretechnik unter dem Begriff *Sichtbarkeit*? (B)

.....

.....

.....

3. Beziehungen
3.1 Was versteht man beim objektorientierten Softwareentwurf unter einer Beziehung? (E)

.....

.....

.....

3.2 Welche Beziehung besteht zwischen einem PKW und den Rädern? (F)

.....

3.3 Ergänzen Sie unten stehendes Klassendiagramm um eine Beziehung zwischen den Klassen. Benennen Sie auch diese Beziehung und geben Sie die Kardinalitäten an. (F, Y)

3.4 Geben Sie 2 Beispiele für eine Komposition an. (F)

.....

.....

4. Generalisierung

4.1 Welchen Vorteil hat die Generalisierung beim objektorientierten Softwareentwurf? (L)

.....

.....

4.2 Was versteht man unter *Polymorphie*? (L)

.....

.....

4.3 Zeichnen Sie das Klassendiagramm für die Klassen: Reise, FlugReise, ZugReise. (M, Y)

4.4 Ergänzen Sie das Diagramm um eine weitere Klasse, welche von der Klasse *Reise* abgeleitet ist. (M)

4.5 Welche der oben genannten Klassen sollte eine abstrakte Klasse sein? (M)

.....

4.6 Was ist der Unterschied zwischen *Überladen* und *Überschreiben*? (L)

.....

.....

5. Softwareentwurf

5.1 Erklären Sie den Unterschied zwischen der objektorientierten Analyse und dem objektorientierten Design! (R)

.....

.....

5.2 Wie gehen Sie bei der Analyse eines Problems vor?

.....

.....

.....

5.3 Wie kann Ihnen dabei das Sequenzdiagramm helfen? (R, X)

.....

.....

5.4 Nennen Sie 2 Designrichtlinien zur Konstruktion des Klassendiagramms. (Q)

.....

.....

5.5 Wozu werden Zustandsdiagramme verwendet? (R)

.....

.....

5.6 Wozu werden *Use Cases* (Anwendungsfälle) verwendet? (U)

.....

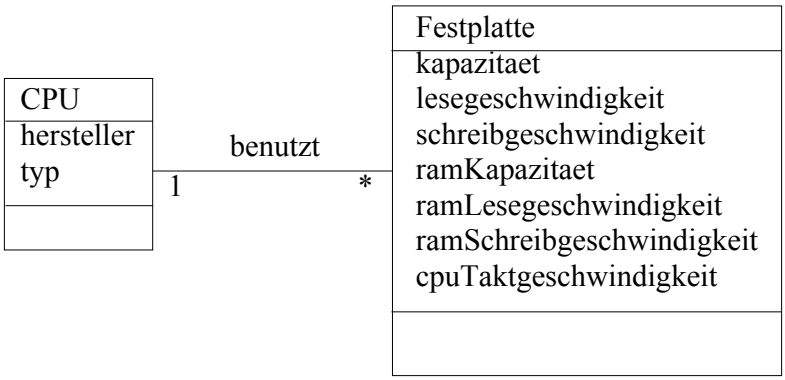
.....

.....

5.7 Klaus möchte ein kleines Programm schreiben, welches eine Band simuliert. In der Band soll es Gitarren, Bässe und Schlagzeuge geben. Jedes Instrument gehört einem Musiker (und wird auch nur von diesem gespielt), es kann aber sein, dass es z.B. mehrere Gitarren in der Band gibt.
Die Musiker können in mehreren Bands mitspielen. (H, O)

a) Entwerfen Sie das passende Klassendiagramm (inkl. benannter Assoziation, Kardinalitäten und abstrakter Klasse)!
b) Tragen Sie die Eigenschaften *tonumfang* und *name* und die Methode *spieleTon()* in die entsprechende Klasse(n) ein!

5.8 Beurteilen Sie das folgende Klassendiagramm. Begründen Sie Ihre Kritik und korrigieren Sie den Entwurf! (I)



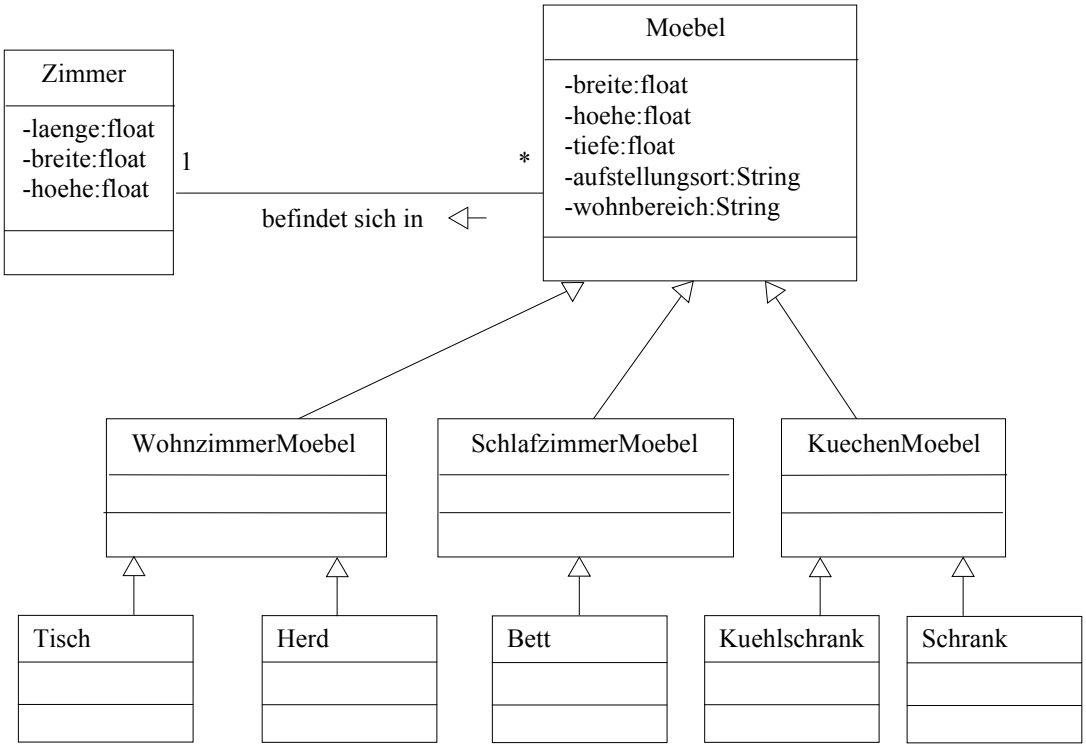
.....

.....

.....

.....

5.9 Beurteilen Sie folgendes Klassendiagramm. Begründen Sie Ihre Kritik! (P)



.....

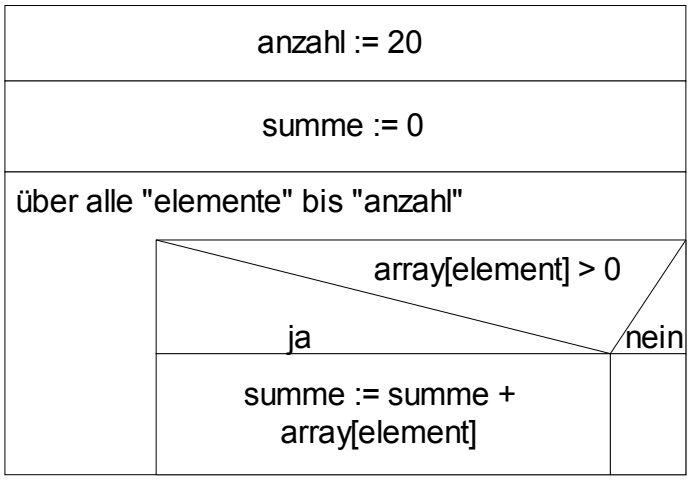
.....

.....

C.6.2 Lösungen

Vortest: Frage 1 – 5.6
Nachtest: Frage 2 – 5.9
Die Lösungen verstehen sich als Musterlösungen und sind teilweise nicht als absolut zu sehen. Die Buchstaben bei den Bewertungsrichtlinien beziehen sich auf die Lernziele (siehe Tabelle 1, Seite 26).

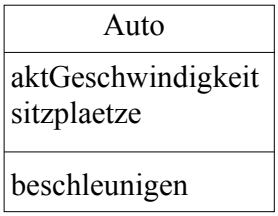
1. Strukturierter Softwareentwurf



- Startbedingung (summe = 0): 2,5 Punkte
- (Kopf)gesteuerte Schleife: 2,5 Punkte
- Auswahlabfrage: 2,5 Punkte
- Richtige Zuweisung für die Summe: 2,5 Punkte

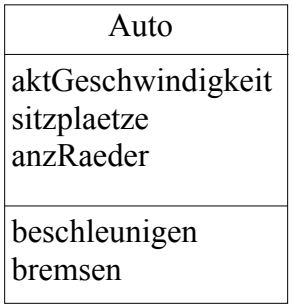
2. Objekt & Klasse

2.1



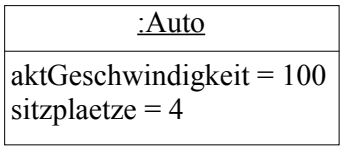
B: korr. Klasse, Eigenschaft, Methode je 3,3 Punkte
Y: normgerechte Klasse nach UML 10 Punkte

2.2



C: Eigenschaft, Methode je 5 Punkte

2.3



C: Kennzeichnen als Objekt der Klasse Auto, Eigenschaften mit Wert, keine Methode, je 3,3 Punkte
Y: normgerechtes Objekt nach UML 10 Punkte

2.4

Klasse: der allgemeine Bauplan, z.B. Auto mit seinen allgemeinen Eigenschaften Farbe, Leistung, etc.
Objekt: ein spezielles, existierendes Auto, wie z.B. das Auto, das Klaus gehört. Es hat die Eigenschaften wie Farbe=rot, Leistung = 50 kW, ...
B: Klasse und Objekt richtig beschrieben je 5 Punkte

2.5

Nicht alle Eigenschaften und Methoden einer Klasse können von einer anderen Klasse verwendet werden. Zugriffsrechte wie public, private und protected regeln die Zugriffsmöglichkeiten. Die entsprechenden Eigenschaften und Methoden sind von außerhalb der Klasse nicht sichtbar.
B: korr. Erklärung 10 Punkte

3. Beziehungen

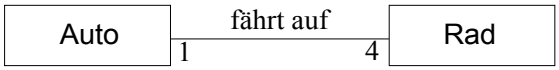
3.1

Eine Beziehung zeigt den Zusammenhang zwischen unterschiedlichen Klassen an.
E: korr. Erklärung 10 Punkte

3.2

Ein PKW fährt auf Rädern.
F: korr. Beziehung 10 Punkte

3.3



F: Bezeichnung, jede korr. Kardinalität je 3,3 Punkte

Y: normgerechtes Diagramm nach UML 10 Punkte

3.4

Hand und Finger, Haus und Zimmer
F: je korr. Beispiel 5 Punkte

4. Generalisierung

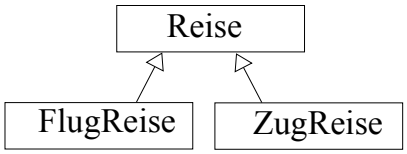
4.1

Übersichtlicheres und besser strukturiertes Design (Informationen sind gebündelt, redundanzfreier); besseres Verständnis der Zusammenhänge; Wiederverwendbarkeit von Code; leichteres Abändern des Codes.
L: Für **einen** Vorteil 10 Punkte

4.2

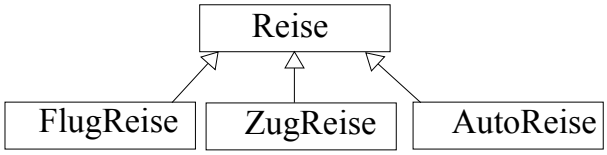
Vielgestaltigkeit. Beim Senden einer Botschaft ist noch nicht bekannt, welche konkrete Methode verwendet wird. Das Objekt selbst kennt nur die korrekte Methode. Hinter einem Methodenaufruf verstecken sich mehrere Methoden.
L: korr. Erklärung 10 Punkte

4.3



M: für die richtige Vererbung je 5 Punkte
Y: normgerechtes Diagramm nach UML 10 Punkte

4.4



M: für eine weitere sinnvoll abgeleitete Klasse 10 Punkte

4.5

Die Klasse *Reise*.
M: für die Wahl der richtigen Klasse 10 Punkte

4.6

Überladen steht für die Verwendung desselben Methodennamens in derselben Klasse aber mit unterschiedlichen Parametertypen. *Überschreiben* meint dagegen die neue Definition einer Methode in einer Subklasse mit demselben Methodennamen und derselben Parameterliste.
L: richtige Erklärung für *Überladen* bzw. *Überschreiben* je 5 Punkte

5. Softwareentwurf

5.1

OOA: **WAS** soll programmiert werden?
OOD: **WIE** soll programmiert werden?
R: richtige Erklärung für OOA bzw. OOD je 5 Punkte

5.2

- suchen nach Klassen (Substantive sind Kandidaten)
- suchen nach Beziehungen zwischen den gefundenen Klassen
- suchen nach Kardinalitäten und evtl. nach Rollen
- suchen nach Attributen in den Klassen (es kann sein, dass für manche Klassen keine Attribute gefunden werden)

keine Punkte!

5.3

Mit seiner Hilfe kann man zeitliche Abläufe übersichtlich darstellen und kann so das Problem analysieren.
R: richtige Erklärung für das Sequenzdiagramm 10 Punkte
X: Sequenzdiagramm bekannt 10 Punkte

5.4

1. Informationsexperte
   Wer ist der Informationsexperte für diese Aufgabe? D.h. wer hat bzw. sollte die Information für diese Aktion haben, wer muss also angefragt werden?
2. geringe Kopplung
   Eine Klasse sollte möglichst wenig Verbindungen zu anderen Klassen aufweisen, so sind bei Änderungen weniger Klassen betroffen.
3. hoher Zusammenhalt

Die Verantwortlichkeit einer Klasse sollte sich möglichst nur auf eine logische Aufgabe beziehen.

4. Erzeuger
   Für die Erzeugung neuer Objekte ist die Klasse zuständig, welche die zu erzeugenden Objekte beinhaltet (Komposition, Aggregation) oder die Informationen zum Start des neuen Objekts enthält oder das zu erzeugende Objekt eng verwendet.

Q: je richtige Richtlinie 5 Punkte (max. 2 Richtlinien; Erklärung ist nicht verlangt)

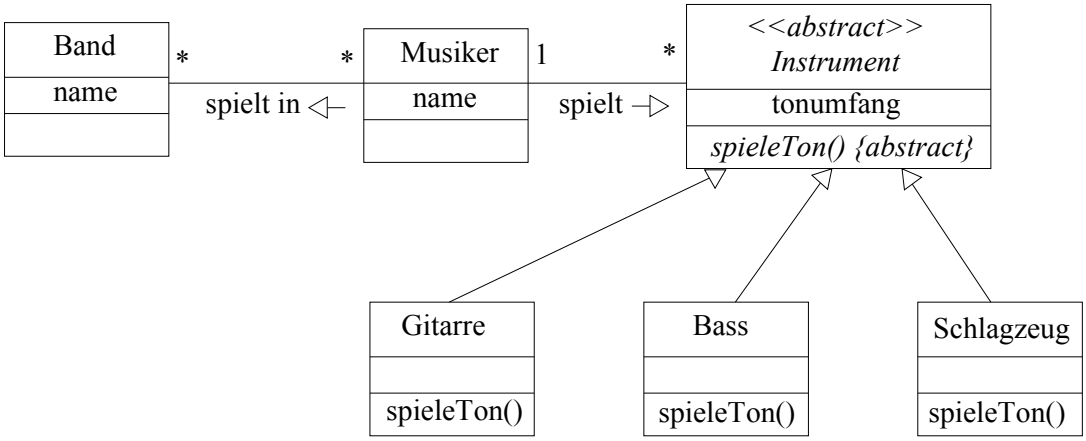
5.5

Zustandsdiagramme werden verwendet, um Probleme mit zustandshaftem Verhalten darzustellen und dafür eine Lösung zu finden.
R: richtige Antwort 10 Punkte

5.6

Use Cases werden verwendet, um die Anforderungen an das Programm zu ermitteln. Sie spielen eine wichtige Rolle in der Problemanalyse.
R: richtige Antwort 10 Punkte

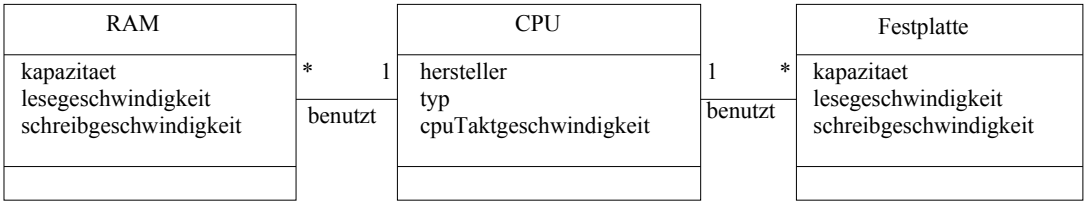
5.7



H: richtige Beziehungen 3 Punkte, Kardinalitäten 3 Punkte, Bezeichnungen und Klassen 3 Punkte, Eigenschaften 1 Punkt
O: richtige Vererbung 5 Punkte, abstrakte Klasse 2,5 Punkte und Methoden 2,5 Punkte

5.8

Die Designrichtlinie *Informationsexperte* ist verletzt, da *cpuTaktgeschwindigkeit* zur CPU gehört. Die Designrichtlinie *hoher Zusammenhalt* ist verletzt, da die Klasse *Festplatte* auch Informationen zum RAM besitzt. Besser ist hier eine eigene Klasse *RAM*.



I: je erkannten Fehler 3,3 Punkte, richtige Korrektur 3,3 Punkte

5.9

- Die Klassen *Moebel*, *WohnzimmerMoebel*, *SchlafzimmerMoebel* und *KuechenMoebel* sollten abstrakt sein.

- Die Klasse *SchlafzimmerMoebel* ist unnötig, da sie nur eine Subklasse besitzt.
- Die Eigenschaften der Klasse *Moebel* müssen protected sein.
- Die Eigenschaften *aufstellungsort* und *wohnbereich* der Klasse *Moebel* sind unnötig, da sie aus der Vererbungsstruktur folgen.
- Die Subklasse *Herd* sollte als Superklasse *KuechenMoebel* und nicht *WohnzimmerMoebel* haben.

P: erkannter Fehler je 2 Punkte

D Softwareübersicht

Ein Ansatz des Unterrichtskonzepts ist die Dekonstruktion (siehe Kapitel 3.7) einer Software. Aus diesem Grund wurde Software entwickelt, welche von den Schülern untersucht werden kann. Anschließend soll diese Software Stück für Stück erweitert werden.
Die Ergebnisse (Spalte Lösung) sollen durch die Schüler selbständig erarbeitet werden. In diesem Sinn sind diese Versionen Musterlösungen. Evtl. kann man diese am Ende als Musterlösungen austeilen, dabei sollte man allerdings nicht den Eindruck erwecken, dass diese die einzigen Lösungen sind. In der Softwaretechnik gibt es immer mehrere Lösungen mit ihren Vor- und Nachteilen. Diese Vor- und Nachteile sollten diskutiert werden!
Die Musterlösungen können aber auch als Ausgangspunkt für den nächsten Schritt verwendet werden.
Die Software wurde mit Java jdk 1.4.2 bzw. mit dem C++-Borland-Builder 3 erstellt.

Version	Lösung	Beschreibung
v1.0		Ausgangszustand, Willi und Siggi erscheinen, können sich zeigen und verstecken, es werden dunkle Punkte gezeichnet, wenn auf die Oberfläche geklickt wird (bei einem *repaint()* verschwinden sie jedoch wieder!).
v1.1	X	Siggi und Willi können nun gehen. Durch einen Mausklick auf die Oberfläche nähern sie sich dem Mausklick.
v1.3	X	Die Methode *gehen()* gibt es nun in zwei überladenen Formen: • gehen(int x,int y), • gehen(double phi).
v1.5	X	Die Methode *setzeAussehen()* wurde durch den Konstruktor ersetzt. Die Eigenschaften des Kobolds – wie z.B. die Augenfarbe – können nur noch bei der Geburt festgelegt werden.
v1.7	X	Es leben nun bis zu 4 Kobolde auf dem Planet.
v1.9	X	Die Kobolde haben einen König.
v1.11	X	Es gibt Männer und Frauen (Vererbung) und die Klasse Kobold ist abstrakt.

Tabelle 30. Die Softwareversionen des Programms "WillisWelt"

Version	Lösung	Beschreibung
v1.1	X	das gesamte Spiel – es sind noch viele Verbesserungen machbar.

Tabelle 31. Die Softwareversionen des Programms "Tic Tac Toe"

Version	Lösung	Beschreibung
v1.1	X	Zeichenprogramm für Kreise, Rechtecke und Dreiecke mit statischem Array
v1.3	X	Nun mit dynamischem Array, d.h. die Anzahl der Graphikobjekte ist nur durch den Speicher begrenzt.
v1.5	X	Die gesamte Graphik kann gelöscht werden.

Tabelle 32. Die Softwareversionen des Programms "Graphik"

Version	Lösung	Beschreibung
v1.1	X	3-stelliges Zahlenschloss (Entwurf über Zustandsdiagramm)
v1.3	X	Zustand wird nun in einer Datei abgelegt, dadurch bleibt auch bei einem Rechnerabsturz der derzeitige Zustand erhalten.

Tabelle 33. Die Softwareversionen des Programms "Zahlenschloss"

Version	Lösung	Beschreibung
v1.1	X	Die Maschine sucht in einem Text nach den möglichen Schreibweisen für Maier (Regulärer Ausdruck: m[ae][iy]er)

Tabelle 34. Die Softwareversionen des Programms "Maier-Suchmaschine"

E Glossar

Aggregation	Teil-Ganzes Beziehung, wobei keine existentielle Abhängigkeit vom Teil zum Ganzen besteht. Engl.: shared aggregation (siehe auch Komposition).
Anwendungsfall	Eine verhaltensorientierte Sequenz von Transaktionen im Dialog mit dem betrachteten System. Angestoßen von einem Benutzer oder einem anderen System (allgemein: etwas außerhalb des betrachteten Systems Stehendem). Er ist Teil des Anforderungsmodells.
Didaktik	Im weitesten Sinne wird hierunter die Theorie des Lehrens und Lernens verstanden. Im engeren Sinne die Theorie der Bildungsinhalte [Bö00]. Im Rahmen dieser Arbeit wird der Begriff im engeren Sinne verstanden, spezieller als die Ziele des Unterrichts. (von gr. didaskein, dt.: lehren, unterweisen)
Entwurfsmuster	Stellt für ein immer wiederkehrendes Entwurfsproblem im Bereich des objektorientierten Softwareentwurfs ein generisches Schema zur Lösung zur Verfügung. Viele Entwurfsmuster finden sich bei [Ga96].
GRASP	General Responsibility Assignment Software Patterns [La02]. Designrichtlinien, um bessere objektorientierte Entwürfe zu erhalten und beurteilen zu können (siehe auch Kapitel 3.4).
GUI	Grafical User Interface. dt.: Graphische Benutzerschnittstelle. Graphische Schnittstelle für die Mensch-Maschine-Interaktion.
Information Hiding	Bei der objektorientierten Programmierung sind die Attribute der Klassen i.d.R. durch Zugriffe von außerhalb der Klasse geschützt (Zugriffsrecht private oder protected). Sie sind meist nur über Methoden der Klasse veränderbar. Dadurch sind die internen Informationen der Klasse nach außen hin versteckt.
Kardinalität	Maß für die Größe einer Menge. Beim objektorientierten Softwareentwurf gibt sie an, wie viele Objekte einer Klasse zu einem Objekt einer anderen Klasse zugeordnet werden.
Komposition	Teil-Ganzes-Beziehung, wobei eine existentielle Abhängigkeit vom Teil zum Ganzen besteht, d.h. wird das Ganze zerstört, wird auch das Teil zerstört Engl.: composit aggregation (siehe auch Aggregation).
Methodik	Der Weg, welcher in Erziehung und Unterricht zur Erreichung bestimmter Ziele verwandt wird [Bö00]. (von gr. methodos, dt.: Weg nach, Weg zu)
Modell	Abbild von etwas, oft unter Weglassen von Details, also im Sinne einer vereinfachten Darstellung [Cl01]. Ein Modell ist ... *im wesentlichen die Abgrenzung eines für den jeweiligen Zweck relevanten Ausschnitts der Erfahrungswelt, die Herausarbeitung seiner wichtigen Merkmale unter Vernachlässigung der unwichtigen sowie seine Beschreibung und Strukturie-*

	rung mit Hilfe spezieller Techniken aus der Informatik . [GI00] (siehe auch Modellieren, Modellbildung)
Modellieren, Modellbildung	Modellieren ist ... *eine abstrahierte Beschreibung eines realen oder geplanten Systems, das die für eine bestimmte Zielsetzung wesentlichen Eigenschaften dieses Systems enthält. Die Erstellung einer solchen Beschreibung nennen wir Modellbildung oder Modellieren.* [Hu99] (siehe auch Modell)
MVC	Model-View-Control Entwurfsmuster [Ad86]. Ein Entwurfsmuster, in dem die Verantwortlichkeiten für die graphische Darstellung, die Steuerung und die Datenablage in Schichten getrennt wird. Jede Klasse im System wird eindeutig einer Schicht zugeordnet.
Objektorientierter Softwareentwurf	Ablauf, um vom Problem zur fertigen Software zu gelangen. Dabei wird von einem objektorientierten Ansatz ausgegangen.
OOA	Objektorientierte Analyse. Im Mittelpunkt steht das Problem selbst, d.h. die Frage: WAS soll gelöst werden?
OOAD	Objektorientierte Analyse und Design (siehe OOA und OOD).
OOD	Objektorientiertes Design. Im Mittelpunkt steht die Lösung, d.h. die Frage: WIE soll das Problem gelöst werden?
OOP	Objektorientierte Programmierung. Im Mittelpunkt steht die Programmierung, d.h. das Codieren in eine Programmiersprache.
Pattern	siehe Entwurfsmuster.
Polymorphie	Wird ein Objekt einer Subklasse erzeugt kann sie auf die Superklasse gecastet werden (sie erscheint nun als Datentyp der Superklasse). Wurde in der Subklasse nun eine Methode der Superklasse überschrieben (siehe Überschreiben), so wird beim Methodenaufruf die Methode der Subklasse verwendet, auch wenn das Objekt als ein Objekt der Superklasse erscheint.
Programmieren	Codieren, d.h. ein Design-Modell mit einer Programmiersprache in ein Programm umsetzen.
Überladen	Beim Überladen unterscheiden sich zwei oder mehrere Methoden innerhalb einer Klasse nur in der Parameterliste. Sie besitzen denselben Methodennamen.
Überschreiben	Beim Überschreiben wird in einer Subklasse eine Methode mit demselben Namen und derselben Parameterliste wie in der Superklasse deklariert.
UML	Unified Modeling Language. Vereinheitlichte Modellierungssprache für die Objektorientierte Analyse und Design (siehe [OMG00]).
Use Case	siehe Anwendungsfall.

F Abbildungsverzeichnis

G Tabellenverzeichnis

H Literaturverzeichnis

[Ab83] Abbott, R.: Program Design by Informal English Descriptions. Communications of the ACM, vol. 26(11), Nov. 1983.

[Ad86] Adams, S.: MetaMethods: The MVC Paradigm. In HOOPLA: Hooray for Object-Oriented Programming Languages. Everette, WA: Object-Oriented Programming for Smalltalk Application Developers Association. Vol 1(4), p. 6, July 1986.

[Ai01] Lehrplan des Fachs Angewandte Informationstechnik, Schulart: Berufliches Gymnasium der dreijährigen Aufbauform – technische Richtung, Profil Informationstechnik; Landesinstitut für Erziehung und Unterricht, Baden-Württemberg (L – 00/3202 02).

[Ba96] Baumann, R.: Didaktik der Informatik. Klett Verlag, Stuttgart, 2. Auflage, 1996.

[Ba97] Baumann, R.: JAVA – Stimulans für den Informatikunterricht. In: Log In, 5/97, S. 19-25.

[Ba99] Balzert, H.: Lehrbuch Grundlagen der Informatik – Konzepte, Notationen in UML, Java, C++, Algorithmik und Software-Technik. Spektrum Akademischer Verlag, Heidelberg, 1999.

[Ba00] Baumann, R.: JAVA im Informatik-Anfangsunterricht – Einführung in die objektorientierte Programmentwicklung (Sekundarstufe II). In: Log In, 1/2000, S. 46-54.

[Ba01] Baumann, R.: Assoziieren und Spezialisieren – Beispiele zum objektorientierten Entwurf in JAVA. Teil 1 in: Log In, 2/2001, S. 10-17; Teil 2 in: Log In, 3,4/2001, S. 67-72.

[Bc94] Booch, Grady: Object-Oriented Analysis and Design – With Applications, Benjamin/Cunnings, Redwood City, 1994; deutsch: Objektorientierte Analyse und Design. Addison-Wesley, Bonn, 1994.

[Be89] Beck, K.; Cunningham, W.: CRC-Methode: A laboratory for teaching OO thinking,. Proceedings of OOPSLA 1998, SIGPLAN Notices, Vol. 24, No. 10, pp 1-6, 1989.

[Be00] Beck, K.; Fowler, M.: Planning Extreme Programming. Pearson Education, Upper Saddle River, 1st Edition, 2000.

[Bl56] Bloom, B.S.: Taxonomie von Lernzielen im kognitiven Bereich. dt. Füner, E., Horn, R.; Beltz Verlag, Weinheim und Basel, 1956. Original: Taxonomy of educational objectives. The classification of educational goals. Handbook I: Cognitive domain. New York, Longmans Green, 1956.

[Bl75] Bloom, B.S.; Krathwohl, D.S.; Masia, B.B.: Taxonomie von Lernzielen im affektiven Bereich. Beltz Verlag Weinheim, 1975. Original: Taxonomy of educational objectives. The classification of educational goals. Handbook II: Cognitive domain .

[Bm01] Baumert, J. et al: PISA 2000, Basiskompetenzen von Schülerinnen und Schülern im internationalen Vergleich; Leske+Budrich, Opladen 2001.

[Bo93] Bortz, J.; Statistik für Sozialwissenschaftler. Springer-Verlag, Berlin, Heidelberg; 4. Auflage, 1993.

[Bo00] Bortz, J.; Lienert, A.; Boehnke, K.; Verteilungsfreie Methoden in der Biostatistik; Springer-Verlag, Berlin, Heidelberg; 2. Aufl., 2000.

[Bö00] Böhm, W.: Wörterbuch der Pädagogik. Kröner, Stuttgart, 15. Auflage, 2000.

[Br99] Breymann, U.: C++ Eine Einführung. Hanser-Verlag, München, 5. Auflage, 1999.

[Br00] Brinda, T,: Objektorientiertes Modellieren – Sammlung und Strukturierung von Übungsaufgaben im Informatikunterricht. In: Log In, 5/2000, S. 39-49.

[Br01] Brinda, T.: Einfluss fachwissenschaftlicher Erkenntnisse zum objektorientierten Modellieren auf die Gestaltung von Konzepten in der Didaktik der Informatik. In: [Ma01], S. 75-86.

[BRJ98] Booch, G.; Rumbaugh, J.; Jacobsen, I.: The Unified Modeling Language User Guide. Addison-Wesley, Reading, Massachusetts, 1999.

[BRJ01] Booch, G., Rumbaugh, J., Jacobsen, I.: The Unified Software Development Process. Addison-Wesley, Boston, 2001.

[Ca63] Cattell,R.B.. Theory of fluid and crystallized intelligence; A critical experiment. Educat. Psycol, 54, 1-22, 1963.

[Ca68] Cattell, R.B.. Are IQ Test intelligent? Psychology today, 10 (1968).

[Cl01] Claus, V., Schwill, S.: Duden Informatik. Bibliographisches Institut & F.A. Brockhaus AG, Mannheim, 2001.

[Ct01] Lehrplan des Fachs Computertechnik, Schulart: Berufliches Gymnasium der dreijährigen Aufbauform – technische Richtung, Profil Informationstechnik; Landesinstitut für Erziehung und Unterricht, Baden-Württemberg (L – 00/3202 03).

[Da68] Dave, R.H.: Eine Taxonomie pädagogischer Ziele und ihre Beziehung zur Leistungsmessung. In: Ingenkamp, K.; Marsolek, T. (Hrsg.): Möglichkeiten und Grenzen der Testanwendung in der Schule. Beltz, Weinheim, 1968.

[De72] Deutscher Bildungsrat, Empfehlungen der Bildungskommission – Strukturplan für das Bildungswesen. Ernst Klett Verlag, Stuttgart, 4. Auflage, 1972.

[Di01] Dietzel, R.; Rinkens, T.: Eine Einführung in die Objektorientierung mit Lego Mindstorms Robotern – Erfahrungsbericht aus dem Unterricht. In: [Ma01] S. 193-199.

[Do87] Dorsch, F. (Hrsg.): Psychologisches Wörterbuch. Huber, Bern Stuttgart, 1987.

[Do00] Doberkat, E.E.: Die Hofzwerge – Ein kurzes Tutorium zur objektorientierten Modellierung. In: Log In, 3,4/2000, S. 71-81.

[Ec00a] Eckel, B.: Thinking in C++, Volume 1: Introduction to Standard C++. Prentice Hall, New Jersey, 2nd Ed., 2000.

[Ec00b] Eckel, B.: Thinking in Java. Prentice Hall, New Jersey, 2nd Ed., 2000.

[El99] Ellen S., Spainhour S., Patwardhan N.: Perl in a Nutshell. dt.: Surmeli D. O'Reilly , Köln, 1999.

[ex99] Microsoft Excel 2000. Microsoft Corporation 1999.

[Fo00] Fowler, M.; Kendall, S.: UML konzentriert – Eine strukturierte Einführung in die Standard-Objektmodellierungssprache. dt. von Mester, A.; Sczittnick, M.; Graw, G.; Addison-Wesley, 2. Auflage, 2000

[Fü99] Füller, K.: Objektorientiertes Programmieren in der Schulpraxis. In: [Sch99] S. 190- 201.

[Ga96] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Entwurfsmuster, Elemente wiederverwendbarer Software. Addison-Wesley, Bonn, 1996.

[GI00] Gesellschaft für Informatik e.V.: Empfehlung der Gesellschaft für Informatik e.V. für ein Gesamtkonzept zur informatischen Bildung an allgemein bildenden Schulen. In: Informatik Spektrum 23 (2000), Nr. 5, S. 378-382.

[Gr01] Gropengießer, H.: Didaktische Rekonstruktion des Sehens: Wissenschaftliche Theorien und die Sicht der Schüler in der Perspektive der Vermittlung. Didaktisches Zentrum Carl von Ossietzky Universität Oldenburg, 2. Auflage, 2001.

[Hb01] Humbert, L.: Informatik lehren – zeitgemäße Ansätze zur nachhaltigen Qualifikation aller Schülerinnen. In: [Ma01], S. 121-132.

[He96] Hermes, A.: OOP im Unterricht – Ein Plädoyer für einen gleitenden Paradigmenwechsel. Teil 1 in: Log In 4/96, S. 29-33; Teil 2 in: Log In 5,6/96, S. 62-67.

[Ho93] Hohlfelder, W.; Bosse, W.: Schulgesetz für Baden-Württemberg – Handkommentar mit Nebenbestimmungen und Sonderteil Lehrerdienstrecht. Boorberg Verlag, Stuttgart, 11. Auflage, 1993.

[Ho96] Hoppe, H.U.; Luther, W.J.: Informatik und Schule – Ein Fach im Spiegel neuer Entwicklungen der Fachdidaktik. In Log In, Heft 1/1996, S. 8-14.

[Hu84] Huberty, C.J.; Issues in the Use and Interpretation of Discriminant Analysis. In: Psychological Bulletin, 1984, Vol. 95, No. 1, 156-171

[Hu99] Hubwieser, P.: Modellierung in der Schulinformatik. In: Log In 1/1999, S. 24-29.

[Hu01] Hubwieser, P.; Didaktik der Informatik: Grundlagen, Konzepte, Beispiele. Springer-Verlag, Berlin, Heidelberg; 1. korrigierter Nachdruck 2001.

[Ie00] Isernhagen, R.: Softwaretechnik in C und C++: Modulare, objektorientierte und generische Programmierung. Hanser-Verlag, München, 2. Auflage, 2000.

[ISO98] ISO/IEC 14882-1998, Information Technology – Programming Languages – C++, Normenausschuss Informationstechnik im DIN, Deutsches Institut für Normung e.V., 10722 Berlin.

[It01] Lehrplan des Fachs Informationstechnik, Schulart: Berufliches Gymnasium der dreijährigen Aufbauform – technische Richtung, Profil Informationstechnik; Landesinstitut für Erziehung und Unterricht, Baden-Württemberg (L – 00/3202 01).

[Ja92] Jacobsen et al: Object-Oriented Software Engineering: A Use Case Driven Approach (ObjectOry). ACM Press, Addison-Wesely, 1992.

[Ja02] Jank, W.; Meyer, H.: Didaktische Modelle. Cornelsen Scriptor, Berlin, 5. Auflage, 2002.

[Jo02] Jobst, F.: Programmieren in Java. Hanser-Verlag, München, 4. Auflage, 2002.

[Ka97] Kattmann, U.; Duit, R.; Gropengießer, H.; Komorek, M.: Das Modell der Didaktischen Rekonstruktion – Ein Rahmen für naturwissenschaftsdidaktische Forschung und Entwicklung. In: Zeitschrift für Didaktik der Naturwissenschaften, 3(3), 3-18.

[Ka01] Kahlbrandt, B.: Software-Engineering mit der Unified Modeling Language. Springer-Verlag, Heidelberg, 2001.

[Kl03] Klieme, E. et al: Zur Entwicklung nationaler Bildungsstandards, Expertiese; Bundesministerium für Bildung und Forschung (BMBF), Referat Publikationen, 10115 Berlin.

[Kn98] Knapp, T.; Fischer, H.: Objektorientierung im Informatikunterricht – Ein didaktisches Konzept zum Einstieg in den Informatik-Unterricht der Sekundarstufe I. In: Log In, 3,4/98, S. 71-76.

[Kö02] König, E.; Zedler, P.: Theorien der Erziehungswissenschaft – Einführung in Grundlagen, Methoden und praktische Konsequenzen. Beltz Verlag, Weinheim, 2. Auflage, 2002.

[Kr52] Kruskal, W.H.; Wallis, W.A. Use of Ranks in one-criterion variance analysis. Journal of the American Statistical Association, 47, 583-621.

[Kr99] Kruchten, Philippe: Der rational unified process : eine Einführung. Addison-Wesley-Longman, München, 1999.

[La02] Larman, Craig: Applying UML and patterns – An Introduction to Object-Oriented Analysis and Design and the Unified Process. Prentice Hall, NewJersey, 2nd Edition, 2002.

[Ma47] Mann, H.B., Whitney, D.R.; On a test of whether one of two random variables is stochastically larger than the other. In: The Annals of Mathematical Statistics, 18, 50-60.

[Ma99] Magenheim, J.; Schute, C.; Hampel, T.: Dekonstruktion von Informatiksystemen als Unterrichtsmethode – Zugang zu objektorientierten Sichtweisen im Informatikunterricht. In [Sch99], S. 149-164.

[Ma01] Magenheim, J.; Keil-Slawik, R.: Informatikunterricht und Medienbildung. 9. GI-Fachtagung Informatik und Schule, INFOS 2001. Gesellschaft für Informatik, Bonn, 2001.

[Me88] Meyer, B.: Object-Oriented Software Construction (Eiffel), Series in Computer Science, Prentice Hall, New York, 1988. deutsch: Objektorientierte Softwareentwicklung. Hanser, London, 1990.

[Mi02] Mittelbach, H.: Einführung in C++. Fachbuchverlag Leipzig, 2002.

[Ne02] Neumann, H. A.: Analyse und Entwurf von Softwaresystemen mit der UML. Hanser-Verlag, München, 2. Auflage, 2002.

[Oe98] Oestereich, B.: Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language. Oldenbourg, München, 4. Auflage, 1998.

[OMG00] OMG Unified Modeling Language Specification. Version 1.3, formal/00-03-01, March 2000; http://www.omg.org.

[Po02] Pospeschill, M.; SPSS für Fortgeschrittene, Durchführung fortgeschrittener statistischer Analysen. Regionales Rechenzentrum für Niedersachsen und Fachrichtung Psychologie der Universität des Saarlandes. 3. Auflage, RRNZ: AWS.STS 3.

[Pu71] Puri, M.L.; Sen, P.K.; Nonparametric Methods in Multivariate Analysis. Wiley & Sons, New York, London, 1971.

[Ru91] Rumbaugh, J. et al: Object-Oriented Modeling and Design. Prentice Hall, 1991; deutsch: Objektorientiertes Modellieren und Entwerfen, Hanser, München, 1994.

[Ru96] Rumbaugh, J.: OMT Insights – Perspectives on Modeling from the Journal of Object-Oriented Programming. Rational Software Corp., SIGS Reference Library, SIGS Books, New York, 1996.

[Sch72] Schott, F., Zur Präzisierung von Lehrzielen durch zweidimensionale Aufgabenklassen, in K.J. Klauer, Lehrzielorientierte Tests; Beiträge zur Theorie, Konstruktion und Anwendung, Pädagogischer Verlag Schwann, Düsseldorf, 1. Auflage, 1972.

[Sch99] Schwill, A. (Hrsg.): Informatik und Schule. Fachspezifische und fächerübergreifende didaktische Konzepte. 8. GI-Fachtagung Informatik und Schule, INFOS99; Springer, Berlin, 1999.

[Sch01] Schubert, S.; Hubwieser, P.; Humbert, L.: Evaluation im Informatikunterricht. In [Ma01] S. 213-215.

[Sp98] Spolwig, S.; Penon, J.: Schöne visuelle Welt? - Objektorientierte Programmierung mit DELPHI und Java. In: Log In, 5/98, S. 40-46.

[Sp99] Spolwig, S.: "Hello World" in OOP – Zum Beitrag von Harro von Lavergne in LOG IN, 18. Jg (1998), Heft 5. In Log In, 5/99, S. 38-42.

[Sp00] Spolwig, S.: Modellieren und Programmieren - "Geld wechseln" als Algorithmiküb ung oder Programmieraufgabe – wann lohnt sich ein Programm zu machen? In: Log In, 2/2000, S. 53-59.

[Sp01] Spolwig, S.: Web-gestützte Softwareprojekte. Teil 1 in: Log In, 6/2000, S. 41-48; Teil 2 in Log In, 1/2001, S. 33-41.

[spss03] SPSS Release 12.0.1 for Windows. SPSS Inc. Headquarters, 233 S. Wacker Drive, 11th floor, Chicago, Illinois 60606, USA. Nov 2003.

[To02] Together 6.1. Borland Software Corporation, Corporate Headquarters 100 Enterprise Way, Scotts Valley, CA 95066-3249, USA. 2002.

[Tu97] Tusche, R.: OOP mit grafischer Benutzeroberfläche – OOP für Client-Server-Anwendungen. In: Log In 3,4/97, S. 27-32.

[Ty71] Tyler, R.W., Basic principles of curriculum and instruction. The University of Chicago Press, Chicago, London, 31. Aufl., 1971.

[UG00] Gesetz über die Universitäten im Lande Baden-Württemberg (Universitätsgesetz – UG) §38, nach http://www.mwk-bw.de/Online_Publikationen/Uni-Gesetz.pdf (Stand 01.02.2000).

[Ve00] Versteegen, G.: Das V-Modell in der Praxis: Grundlagen, Erfahrungen, Werkzeuge. dpunkt-Verlag, Heidelberg, 1. Auflage, 2000.

[vl98] von Lavergne, H.: Thesen zur aktuellen Orientierungslosigkeit – welche Schwerpunkte kann die Informatik-Ausbildung künftig haben? In: Log In, 5/98, S. 19-23.

[vl99] von Lavergne, H.: Visuelle Welt – Schön, The missing link. In: Log In, 5/99, S. 43-49.

[We98] R.H. Weiß. Grundintelligenztest Skala 2, CFT 20. Hogrefe, Verlag für Psychologie, Göttingen, Bern, 1998, 4. Auflage.

[We01] Weigend, M.: Objektorientiertes Modellieren von Animationen mit Flash. In: Log In, 5,6/2001, S. 54-57.

[Wi] Wiedemer, L., Zwosta, M.: Objektorientierte Softwareentwicklung mit Java. Script zur Fortbildung, KHS Donaueschingen.

[Za85] Zahradnik et al: Der goldene Naturführer: Pflanzen und Tiere Mitteleuropas, 50 Jahre Kosmos-Naturführer. Franckh, Stuttgart, 1985.

[Zw85] Zwick, R.; Nonparametric One-Way Multivariate Analysis of Variance: A Computional Approach Based on the Pillai-Bartlett Trace. In: Psychological Bulletin, 1985, Vol. 97, No.1, 148-152.