

# Projektaufgaben in Anfängerveranstaltungen: ein Mittel zur Förderung eines objektorientierten Programmierstils

Christoph Kreitz

Fachgebiet Intellektik, Fachbereich Informatik,

Technische Hochschule Darmstadt

Alexanderstraße 10, D-64283 Darmstadt

## Zusammenfassung

Im Informatik-Grundstudium der TH Darmstadt haben wir im Wintersemester 1993/94 damit begonnen, die objektorientierte Sicht auf die Programmierung vorrangig vor den algorithmischen Programmierkonzepten zu behandeln, um Studenten schon zu Beginn ihres Studiums an den Entwurf größerer Softwaresysteme und die hierfür nötigen Arbeitsweisen heranzuführen. Um dieses Lehrkonzept zu unterstützen und die Vorteile eines objektorientierten Programmierstils herauszustellen, wurde in den Übungen der Erstsemesterveranstaltung eine projektartige Leitaufgabe “Flugverkehrverwaltung für Europa” behandelt, welche aufgrund ihres Schwierigkeitsgrades eine Teamarbeit notwendig machte, wegen ihrer Thematik ein objektorientiertes Vorgehen nahelegte, und für die Studenten interessant genug war, um sie immer wieder aufzugreifen, wenn es darum ging, neu erworbenes Wissen zur Anwendung zu bringen.

Nach der Vorstellung der beabsichtigten Lernziele und der Konzeption der Projektaufgabe werden wir anhand einer nachträglichen Analyse des tatsächlichen Lernerfolgs Chancen, Schwierigkeiten und Risiken von Projektaufgaben in Anfängerveranstaltungen diskutieren und Empfehlungen ansprechen, wie derartige Projekte erfolgreich zur Unterstützung des Lernens im Grundstudium Informatik eingesetzt werden können.

## 1 Objektorientierung in der Anfängerveranstaltung

Der viersemestrige Veranstaltungszyklus “Grundzüge der Informatik” bildet den Kernbestandteil des Grundstudiums Informatik an der TH Darmstadt. Er soll die Studenten in die Fachsystematik einführen und Ihnen die fundamentalen Methoden und Kenntnisse der Informatik vermitteln. Schwerpunkt des ersten Semesters ist das Thema *Programmierung* mit höheren Programmiersprachen, während im zweiten Semester die maschinennahe Programmierung und im dritten

Semester die wichtigsten Algorithmen- und Datenstrukturen im Vordergrund stehen. Eine Auseinandersetzung mit den theoretischen Grenzen von Algorithmen findet schließlich im vierten Semester statt. Aufbauend auf diesen Grundlagen bietet das Hauptstudium Informatik eine große Vielfalt anwendungsbezogener Lehrveranstaltungen an, insbesondere auch eine zweisemestrige Vorlesung und ein ausführliches Praktikum zum Software-Engineering, in der die Praxis der Programmierung intensiv vertieft werden kann.

In dem Einführungszyklus kommt der Anfängerveranstaltung eine besondere Rolle zu. Sie soll den Studenten einen ersten Kontakt mit *wissenschaftlichen* Denkformen und *systematischen* Vorgehensweisen der Programmierung bieten und ihnen ermöglichen, praktische Erfahrungen bei der Realisierung von Programmierproblemen zu sammeln. Dies bedeutet, daß die Studenten lernen, mit *formalen Systemen* wie Logik, abstrakten Maschinen und Programmiersprachen umzugehen, Systeme bescheidenen Umfangs zu *entwerfen*, den Entwurf in einer problemorientierten Programmiersprache zu *realisieren* und schließlich die *Qualität* des Entwurfs (Strukturierung) und der Realisierung (Zuverlässigkeit, Effizienz) zu *analysieren*.

Darüberhinaus soll die Veranstaltung aber auch eine Brücke zwischen Schule und Universität darstellen. Unterschiedliche Vorkenntnisse müssen ausgeglichen und falsche Prägungen<sup>1</sup> revidiert werden. Die Studenten müssen an eine andere, wesentlich selbständigere Art des Lernens herangeführt werden, als sie es bisher gewohnt waren. Sie sollen Selbständigkeit, Kreativität, Teamfähigkeit und Verantwortungsbewußtsein entwickeln. Auch auf ihre Persönlichkeitsbildung als erwachsene Menschen hat die Gestaltung der Erstsemesterveranstaltungen einen nicht unbedeutenden Einfluß.

<sup>1</sup>Erfreulicherweise besitzt ein Großteil der Studenten bereits Erfahrungen im Umgang mit Computern, obwohl dies keine Voraussetzung für das Informatik-Studium ist. Die meisten dieser Studenten haben sich leider jedoch auch eine “Hackermentalität” angeeignet und sehen die Notwendigkeit einer systematischen Vorgehensweise beim Lösen von Programmierproblemen nicht ein, solange sie das Problem noch zu überschauen glauben.

Um diese fachlichen und überfachlichen Lehrziele zu erreichen, gliedern sich die “Grundzüge der Informatik I” — wie auch die Veranstaltungen des zweiten und dritten Semesters — in eine Vorlesung, eine Kleingruppenübung und ein begleitendes Praktikum mit einem Gesamtumfang von 8 Semesterwochen.

- In der *Vorlesung* stehen die wissenschaftlichen Aspekte der Programmierung im Vordergrund. Es werden die grundlegenden Konzepte und Denkformen besprochen, die nötig sind, um eine hohe Qualität bei den entwickelten Programme sicherzustellen. Zur Illustration der vorgestellten Konzepte wird üblicherweise eine konkrete problemorientierte Programmiersprache verwendet, von der aber nur die didaktisch relevanten Aspekte zur Sprache kommen. Die Vermittlung von Details, die nur für eine effiziente Implementierung konkreter Problemstellungen erforderlich sind, steht dagegen im Hintergrund.
- In den *Übungen* sollen sich die Studenten durch eigene Aktivitäten mit dem Thema Programmierung auseinandersetzen, wobei auch hier die grundsätzlichen Denkweisen wie Systematik, Entwurf und Analyse im Vordergrund stehen. Sie sollen insbesondere lernen, Aufgaben in kleinen Gruppen (ca. 3–7 Personen) selbständig und kreativ zu lösen und die Qualität dieser Lösungen ausdiskutieren. Eine Übungsgruppe besteht aus mehreren solchen Kleingruppen, denen ein Tutor beratend zur Seite steht.
- In dem die Veranstaltung begleitenden *Praktikum* sollen die Studenten trainieren, kleinere Programmsysteme im Detail auf dem Rechner zu realisieren. Hierbei sollen sie auch die Feinheiten einer Programmiersprache kennenlernen, die für eine konkrete Umsetzung abstrakt entworfener Lösungen erforderlich sind, und lernen, die hierbei auftretenden praktischen Hindernisse zu überwinden. Dies geschieht zunächst anhand von sehr einfachen Problemstellungen und mündet zum Ende des Semesters in einer größeren Programmieraufgabe, deren Lösung mehrere Wochen in Anspruch nimmt.

Obwohl Vorlesung, Übung und Praktikum unterschiedliche Teilziele verfolgen und z.T. getrennt voneinander organisiert werden, bilden diese drei Teilveranstaltungen zusammen eine Einheit, welche den Studienanfängern eine Einführung in die wichtigsten Aspekte der Programmierung bietet. Sie soll vor allem dafür sorgen, daß sich die Studenten frühzeitig die richtigen Denkweisen und Methoden angewöhnen, die beim Umgang mit “realen” Problemen unbedingt erforderlich sind. Nichtsdestotrotz steht der Einführungscharakter bei dieser Veranstaltung im Vordergrund; als Ersatz für eine Software-Engineering Vorlesung ist sie nicht gedacht.

Entsprechend dem Stand der Technik in der Programmierung wird seit dem Wintersemester 1991/92 die objektorientierte Sicht und das “Programmieren im Großen” in den Vordergrund gestellt. Für das Praktikum wurde die Programmiersprache Eiffel gewählt, da hier die Objektorientierung nicht aufgepropft wirkt und das in Eiffel enthaltene Typkonzept aus didaktischer Sicht am überzeugendsten wirkt. Diese Entscheidung wurde von den Professoren des Fachbereichs und den betroffenen Studenten im wesentlichen begrüßt (siehe [Henhapl & et. al, 1994]), so daß Eiffel mit wachsendem Erfolg im Grundstudium Informatik der TH Darmstadt eingesetzt wird.

In den ersten beiden Zyklen versuchte die Ausbildung, der historischen Entwicklung von Programmiersprachen zu folgen und an vorhandene Programmierkenntnisse anzuknüpfen. Es wurden zunächst die einfachen Elemente von Programmiersprachen wie Ausdrücke und algorithmische Strukturen besprochen und erst später auf objektorientierte Konzepte zur Strukturierung und Lösung komplexerer Probleme eingegangen. Durch diese “bottom-up” Vorgehensweise kamen die spezifischen Denkweisen der objektorientierten Programmierung allerdings erst gegen Ende des ersten Semesters zur Sprache. Viele Studenten empfanden Eiffel deshalb als unnötig kompliziert und haben ihre Stärken bei der Strukturierung und Implementierung von Softwaresystemen nicht wirklich verstanden (siehe [Hornecker, 1995, Kapitel 1.3.3]).

Im dritten Zyklus haben wir daher damit begonnen, die objektorientierte Sicht auf die Programmierung von Anfang an konsequent in den Vordergrund zu stellen und den systematischen Entwurf von Software sowie die zugehörigen Sprachkonzepte von Eiffel – Objekte, Klassen, Kontrakte, Vererbung etc. – vor den konventionellen algorithmischen Konzepten zu behandeln (siehe [Kreitz, 1994]). Hierdurch sollte nicht nur eine gewisse Chancengleichheit zwischen Studienanfängern mit und ohne Computerkenntnisse erreicht sondern auch verhindert werden, daß sich die Studenten einen prozedurorientierten Programmierstil aneignen, den sie später nicht wieder ablegen können. Stattdessen sollte sie ein “top-down” Ansatz schon zu Beginn ihres Studiums an den Entwurf von Softwaresystemen realistischer Komplexität und die hierfür nötigen Arbeitsweisen heranführen.

Da sich die Studenten bei dieser Vorgehensweise zunächst mit formalen Spezifikationen und den abstrakteren Strukturierungskonzepten auseinandersetzen müssen, bevor sie ihre Kenntnisse zur Implementierung eigener Algorithmen auf dem Computer verwenden können, ist es nötig, das Lehrkonzept durch eine entsprechend veränderte Konzeption der Übungen und des Praktikums zu unterstützen. Anstatt die Studenten zuerst ausschließlich mit kleineren Programmieraufgaben “herumspielen” zu lassen, die sie alleine zu lösen haben und in wenigen Stunden implementieren können, bietet es sich nun an, sie *von vornehe-*

*rein* auch mit einem Programmierproblem einer realistischen Größenordnung zu konfrontieren, dessen Bearbeitung sich über eine längere Zeit hinzieht und von einer einzelnen Person gar nicht zu bewältigen ist. Auf diese Art können die Studenten die Vorteile der objektorientierten Programmierung gegenüber einem schnellen "Hack" erkennen und lernen frühzeitig, an große Projekte heranzugehen, im Team zu arbeiten, Softwarekomponenten wiederverwendbar und zuverlässig zu gestalten und bei der Programmentwicklung systematisch vorzugehen. Die Erfahrungen der Vorjahre und Gespräche mit der Fachschaft ließen zudem erwarten, daß eine derartige Projektaufgabe für Studenten erheblich interessanter und motivierender sein würde als die Erstellung von meist als nutzlos und unrealistisch empfundenen "Spielprogrammen".

In Ergänzung zu den konventionellen Übungs- und Praktikumsaufgaben wurde daher als Leitaufgabe für die Veranstaltung des Wintersemesters 1993/94 ein Projekt "Flugverkehrsverwaltung für Europa" gestaltet, welches innerhalb des Semesters in mehreren Phasen bearbeitet und schließlich im Rahmen des Praktikums zur Implementierung gebracht werden sollte. Diese Aufgabe legte aufgrund ihrer Thematik ein objektorientiertes Vorgehen nahe und machte wegen ihres Schwierigkeitsgrades eine Teamarbeit unbedingt erforderlich. Andererseits war sie von der Problemstellung her leicht vorzustellen und zumindest im Prinzip lösbar. Zudem war sie interessant genug, um immer wieder darauf zurückzugreifen, wenn es darum ging, neu erworbenes Wissen aus der Vorlesung zur Anwendung zu bringen.

In dieser Arbeit wollen wir über unsere Erfahrungen mit dem Einsatz derartiger Projektaufgaben in einer Einführungsveranstaltung berichten und diskutieren, inwieweit diese dazu genutzt werden können, ein objektorientiertes Denken bei der Lösung von Programmierproblemen zu fördern. Im Abschnitt 2 werden wir die Konzeption für die Durchführung der Leitübung "Flugverkehrsverwaltung für Europa" und ihre Einbettung in das Gesamtkonzept der Veranstaltung vorstellen. Abschnitt 3 präsentiert eine rückwirkende Analyse der Vorteile und Probleme der tatsächlich durchgeführten Projektaufgabe. In Abschnitt 4 werden wir schließlich einige Maßnahmen ansprechen, einen erfolgreichen Einsatz derartiger Projektaufgaben bei der Vermittlung objektorientierter Grundtechniken im Informatik-Grundstudium unterstützen können.

## 2 Die Projektaufgabe "Flugverkehrsverwaltung"

Oberstes Ziel bei der Gestaltung der Leitübung war es, eine Aufgabenstellung zu wählen, die einerseits das typische Aufgabenfeld eines Informatikers verdeutlicht, andererseits aber von der Problemstellung her leicht vorzustellen ist, so daß eine Lösung auch von Pro-

grammieranfängern entworfen werden kann. Um die Projektaufgabe für die Studenten interessant und realistisch genug zu gestalten, hatten wir ein Szenario entworfen, in das wir die Studenten sich hineinzusetzen konnten.

Stellen Sie sich vor, Ihre gesamte Übungsgruppe, also etwa 20 Leute, wären eine Softwarefirma, die den Auftrag übernommen hätte, ein Programm für die Verwaltung des gesamten Flugverkehrs über Europa zu schreiben. Das Programm soll bei Änderungen der Wetter- oder Verkehrslage an einem oder mehreren Flughäfen die aktuellen Flugpläne automatisch so modifizieren, daß zu jeder Zeit ein weitestgehend reibungsloser Ablauf des Flugverkehrs gewährleistet ist. Die ausgelösten Aktionen sind in für spätere Kontrollmöglichkeiten in einem Protokoll festzuhalten.

Hierzu haben Sie von ihrem Auftraggeber bisher nur eine knappe Liste von nicht immer sehr präzise formulierten Rahmenbedingungen und Mindestanforderungen bekommen, die möglicherweise auch nicht ganz vollständig oder widerspruchsfrei sind. Ihre Firma hat sich verpflichtet, binnen eines halben Jahres ein Programmsystem zu implementieren, welches die geforderten Leistungen erfüllen kann.

Es war für die Studenten leicht einzusehen, daß diese Aufgabe viel zu komplex war, um eine Lösung durch eine Einzelperson zuzulassen, und deshalb auf mehrere Teilgruppen verteilt werden mußte, die sich untereinander abzustimmen aber unabhängig zu arbeiten hatten. Die Notwendigkeit, bei dem Entwurf der Lösung auf *Qualitätskriterien* wie Modularität und Zuverlässigkeit zu achten, war damit ebenso klar wie die Tatsache, daß nur eine systematische Herangehensweise zum Ziel führen konnte. Auch ein gewisses Gefühl der Überforderung durch die gestellte Aufgabe war durchaus beabsichtigt. Es wurde den Studenten jedoch versichert, daß sie mit den Konzepten, die sie im Laufe der Veranstaltung kennenlernen würden, und ihrem gesunden Menschenverstand durchaus in der Lage wären, bis zum Ende des Semesters eine Lösung zu erstellen.

Ohne eine gezielte Anleitung wäre eine derartigen Aufgabenstellung durch Anfänger natürlich kaum zu lösen gewesen. Es war daher vorgesehen, auf etwa jedem zweiten Übungsblatt Aufgaben zu stellen, die zu einer Weiterbearbeitung des Projektes FEVS (Flughafen Ereignis VerarbeitungsSystem) anregen sollten. Die Studenten sollten schrittweise die informale Spezifikation präzisieren, in eine formale Spezifikation umwandeln, Objekte bestimmen und als Klassen beschreiben und zum Schluß im Rahmen des Praktikums gruppenweise Teile des gesamten Systems implementieren. Zudem wurde die Problembeschreibung im Laufe der Zeit verfeinert und erweitert, um neue konzeptionelle Aspekte bearbeiten zu können, die zu früheren

Zeitpunkten ein Übermaß an Informationen bedeutet hätten.

Die einzelnen Phasen der Leitübung, die wir im folgenden vorstellen wollen, richteten sich nach den konkret präsentierten Inhalten der Vorlesung, die in 4 Hauptbereiche aufgeteilt war.

1. *Einführung, Qualitätskriterien, methodische Übersicht*
2. *Formale Sprachbeschreibungen – Logik als Spezifikationsprache*
3. *Objektorientierte Konzepte*
  - Klassen & Objekte, Routinen, Datenkapselung, Generizität, Kontrakt-Modell, Vererbung
  - Methodik des Systementwurfs
4. *Algorithmische Konzepte*
  - Korrektheit & Verifikation, Algorithmenstrukturen, Ausdrücke
  - Methodik des Algorithmenentwurfs
  - Ethik und Verantwortung des Informatikers

Die Vorlesung wurde durch ein ausführliches Skriptum [Kreitz, 1994] ergänzt, welches sich an die Bücher von Meyer [Meyer, 1988, Meyer, 1992] und Gries [Gries, 1981] anlehnte. Zusätzlich zu diesen fachlichen Inhalten wurde etwa 6 Wochen nach Semesterbeginn im Rahmen der Vorlesung eine besondere Veranstaltung zum Thema “Lernen im Grundstudium” (in Zusammenarbeit mit der hochschuldidaktischen Arbeitsstelle der TH Darmstadt) durchgeführt, um den Studenten die Umstellung von schulischem Lernen zum universitären Lernen zu erleichtern.

Im Rahmen der Übungsstunden, in denen nicht am Projekt gearbeitet wurde, wurden kleinere Aufgaben besprochen, die vor allem diejenigen Aspekte beleuchteten, die im Rahmen des Projekts nicht vertieft werden konnten oder einer separaten Einübung bedurften.

### **Phase 1: Natürlichsprachliche Problembeschreibung**

Die erste Übungsstunde sollte dazu genutzt werden, die Studenten mit der Aufgabenstellung, die in Abbildung 1 verkürzt dargestellt ist, vertraut zu machen. Sie sollten versuchen, die Vorgaben des “Auftraggebers” zu präzisieren, offensichtliche Lücken zu schließen und als Endergebnis ein informelles “Pflichtenheft” erstellen, welches Grundlage für eine spätere Realisierung sein könnte. Darin sollten neben den Anforderungen an das Programm auch die Annahmen über Rahmenbedingungen fixiert werden, die in der Aufgabenstellung zwar nicht genannt waren, aber für einen korrekten Ablauf der Flugverkehrsverwaltung nötig oder sehr sinnvoll sind. Zur Inspiration wurden einige solcher Bedingungen bereits auf dem Aufgabenblatt genannt wie zum Beispiel

- *Jeder Flughafen hat einen eindeutigen Namen.*

- *Jeder Flug hat einen Flugzeugtyp, der in der Menge der Flugzeugtypen enthalten ist.*
- *Jeder Flug hat einen Flugzeugtyp, dessen maximale Flugzeit größer als die Flugdauer ist.*

Da in der Vorlesung bis zu diesem Zeitpunkt nur eine grobe Einführung in die Programmierung gegeben worden und die wichtigsten Qualitätskriterien für Software angesprochen worden waren, kam es in dieser Übung vor allem darauf an, den gesunden Menschenverstand einzusetzen und Anforderungen sowie Rahmenbedingungen auf Plausibilität, Vollständigkeit und Widerspruchsfreiheit zu prüfen.

### **Phase 2: Formale Spezifikation**

In den Wochen zwischen Phase 1 und 2 wurden in der Vorlesung formale Sprachen und die Prädikatenlogik behandelt. Aufgabe der zweiten Phase war es nun, das in Phase 1 erstellte Pflichtenheft in der Sprache der Prädikatenlogik geeignet zu formalisieren und ggf. um weitere Bedingungen zu ergänzen, die in der Zwischenzeit als notwendig oder sinnvoll erkannt wurden.

Diese formale Spezifikation sollte später als Ausgangspunkt für den Entwurf der Systemstruktur und Vor- und Nachbedingungen von Routinen dienen. Zudem sollte verstanden werden, daß die Prädikatenlogik zwar ein sinnvolles Handwerkzeug bei der Formalisierung anbietet und Zweideutigkeiten in der Anforderungsliste zu vermeiden hilft, aber durchaus ein gewisses Spektrum möglicher Formulierungen der Spezifikation offenläßt. Bedingt durch die Komplexität der Aufgabenstellung wurde die formale Spezifikation schon jetzt relativ aufwendig und es war sinnvoll, bereits in dieser Phase eine gewisse Aufgabenverteilung auf Teilgruppen vorzunehmen.

### **Phase 3: Grobentwurf des FEVS Systems**

Nachdem in der Vorlesung die Konzepte abstrakter Datentyp, Klasse, Objekt, Routine, Typisierung sowie die elementarsten Operationen auf Klassen behandelt worden war, konnte ein erster Grobentwurf des FEVS Systems erstellt werden, welcher die Namen und Typen von Klassen und Routinen fixierte. Die Studenten wurden daher damit beauftragt, die in der Problemstellung erwähnten realen Objekte und die für deren Verarbeitung relevanten Aspekte und Abhängigkeiten zu bestimmen. Zur Beschreibung sollten dann Eiffel-Klassen benutzt werden. Operationen sollten durch Eiffel-Routinen dargestellt werden, von denen zunächst nur die Namen und Typen von Argumenten und Ergebnissen festzulegen waren. Das gewünschte Verhalten der Routinen sollte in einem Kommentar festgehalten werden. Routinen, die sich ausschließlich mit einer einfachen Sequenz von Grundoperationen auf Klassen realisieren ließen, sollten vollständig ausprogrammiert werden.

Die Flugverkehrsverwaltung berücksichtigt *Städte, Flughäfen, Flugzeugtypen, Flüge* und *Ereignisse*.

**Städte** haben einen Namen und eine Anzahl von Flughäfen.

**Flughäfen** haben einen Namen, eine Stadt in deren Nähe sie lokalisiert sind und eine Anzahl von abfliegenden und ankommenden Flügen. Der Zustand eines Flughafens kann *clear, fog, snow, traffic* oder *heavy traffic* sein. Ein Flughafen hat eine Kapazität, welche besagt, wieviele Flüge abgewickelt werden können, ohne daß es zu Verspätungen kommt, und eine Menge von Ausweichflughäfen samt ihrer Entfernung. Es ist bekannt, ob Landungen bzw. Starts bei Nebel möglich sind. Schließlich ist jedem Flughafen eine Folge von bisherigen Ereignissen zugeordnet.

**Flüge** haben eine Flugesellschaft, Flugnummer, Flugzeugtyp, Start- und Zielflughafen, Abflug- und Ankunftszeit und einen Status (*regular, delayed, late, diverted, cancelled* oder *forced landing*).

**Flugzeugtypen** haben eine Bezeichnung, eine Passagierkapazität, eine maximale Reichweite (in Flugstunden) und eine Liste von Flügen, die von Flugzeugen dieses Typs durchgeführt werden. Es ist bekannt, ob Flugzeuge dieses Typs über eine Nebellandevorrichtung verfügen.

**Ereignisse** sind wetterbedingte Veränderungen des Zustands eines Flughafens. Sie werden durch die Vorkommenszeit, den betroffenen Flughafen und den erzeugten Zustand (*clear, fog, oder snow*) beschrieben. Ereignisse haben folgende direkte und indirekte Auswirkungen auf Flughäfen und Flugpläne:

*snow*: Flüge, die in einem Flughafen landen sollen, der zum Zeitpunkt der Ankunft verschneit ist, werden zum nächsten Ausweichflughafen umgeleitet, der *clear* ist und für den Flugzeugtyp erreichbar ist. Die Ankunftszeit dieser Flüge wird entsprechend der Entfernung zu diesem Ausweichflughafen verspätet und der Flug gilt als *diverted*. Erfüllt keiner der Ausweichflughäfen die o.g. Bedingungen, dann werden die betroffenen Flüge zu dem Flughafen zurückgeschickt, von dem aus sie gestartet sind und die Ankunftszeit entsprechend der bisherigen Flugzeit berechnet. Flüge, die innerhalb von 5 Stunden nach dem Einsetzen des Schnees hätten starten sollen, erhalten den Status *cancelled*.

*fog*: Flüge, die in einem Flughafen landen sollen, auf dem zur Ankunftszeit Nebel herrscht, werden um 30 Minuten verspätet, wenn keine Landung bei Nebel möglich ist. Sie erhalten den Status *delayed*, wenn der bisherige Status *regular* ist bzw. *late*, wenn der bisherige Status *delayed* ist. Löst sich der Nebel vor der aufgeschobenen Ankunftszeit dieser Flüge nicht auf, so werden sie zum nächsten Ausweichflughafen umgeleitet, wobei die gleichen Regeln wie beim Auftreten von Schnee gelten.

Der Abflug von Flügen, die in einem nebligen Flughafen ohne Nebellandevorrichtung starten sollen, wird 30 Minuten verzögert. Ihre Ankunft wird um 30 Minuten verspätet, wenn die ursprüngliche Flugzeit weniger als 4 Stunden beträgt, und ihr Status entsprechend verändert. Löst sich der Nebel innerhalb dieser 30 Minuten nicht auf, so wird der Abflug um weitere 30 Minuten verzögert.

*traffic*: Wird die reguläre Kapazität eines Flughafens um bis zu 50% überschritten und ist der Status der Flughafens *clear*, so erhält der Flughafen den Status *traffic*. Ankommende Flüge werden um 30 Minuten verspätet und ihr Status entsprechend verändert. Ebenso werden startende Flüge um 30 Minuten verzögert, wobei die gleichen Regeln wie bei nebelbedingten Abflugverzögerungen gelten.

*heavy traffic*: Wird die reguläre Kapazität eines Flughafens um mehr als 50% überschritten und ist sein Status *clear* oder *traffic* so erhält der Flughafen den Status *heavy traffic*. Flüge, die innerhalb von einer Stunde nach Beginn dieses Zustands landen sollen, werden um 1 Stunde verspätet. Sie erhalten den Status *late*, wenn der bisherige Status *regular* oder *delayed* ist. Flüge, die später als 1 Stunde nach Beginn des *heavy traffic*-Status landen sollen und nicht bereits *diverted* sind, werden zum nächsten geeigneten Ausweichflughafen umgeleitet. Sie erhalten den Status *diverted* und ihre Verspätung wird entsprechend bestimmt. Flüge, die von einem Flughafen starten sollen, der sich im *heavy traffic*-Status befindet, erhalten den Status *late*. Ihr Abflug wird um eine Stunde verspätet. Ihre Ankunft wird um 30 Minuten verspätet, wenn die Flugzeit mehr als 4 Stunden beträgt, andernfalls um 1 Stunde.

Flüge, deren Flugzeit aufgrund einer Verspätung oder Umleitung die maximale Flugzeit ihres Flugzeugtyps überschreitet, landen im Flughafen, in dem die Verspätung oder Umleitung ausgelöst wurde, und erhalten den Status *forced landing*. Ihre Abflug- und Ankunftszeiten bleiben unverändert.

**Aufgabenstellung:** Gegeben sei als Szenario eine Menge von Städten, Flughäfen, Flugzeugtypen, Flügen und Ereignissen, die innerhalb eines Zeitintervalls von 24 Stunden vorkommen. Entwerfen und implementieren Sie ein Programmsystem, das alle Veränderungen der Flug- und Flughafenzustände entsprechend der obigen Problembeschreibung berechnet und protokolliert.

Abbildung 1: Problemstellung für das Flugverkehrsverwaltungsprogramm (Kurzfassung)

In dieser Phase sollten die einzelnen Untergruppen ihre Entwürfe zunächst unabhängig voneinander entwickeln und dann gemeinsam in der Übungsstunde diskutieren. Da sich die Modellierung weitestgehend an der ihnen vertrauten Realität orientieren konnten, war zu erwarten, daß auch Studenten ohne Programmiererfahrung einen solchen Entwurf erstellen konnten (was ihnen bei einem "konventionellen" Entwurf wesentlich schwerer fallen würde). Außerdem konnten die Studenten die in dieser Phase die Benutzung von Objekten und Routinen erlernen, ohne dafür Details der Sprache Eiffel kennen zu müssen.

#### **Phase 4: Klassenfeinspezifikation**

Die Phase 4 folgte unmittelbar auf die dritte Phase. Es ging darum, daß in der Vorlesung besprochene Vertragskonzept zwischen Kunden- und Lieferantenklassen umzusetzen und die Grobspezifikation von FEVS durch Vor- und Nachbedingungen für einzelne Routinen sowie durch Klasseninvarianten zu verfeinern. Für diesen Zweck mußten auch weitere, in der ursprünglichen Problembeschreibung nicht explizit genannte Klassen wie z.B. eine Klasse für die Uhrzeit spezifiziert werden.

Geplant war auch, in dieser Phase bereits festzulegen, welche Teilgruppe später welche der spezifizierten Klassen implementieren sollte, so daß der Vertragsgedanke wirklich zu einem *Aushandeln* sinnvoller Vor- und Nachbedingungen führen sollte. Den Studenten sollte klar werden, daß sie mit der Fixierung dieser Bedingungen auch die Verantwortung übernahmen, diese später zu erfüllen, und sich andererseits bei der Implementierung darauf verlassen dürfen, daß andere Teilgruppen, deren Routinen sie benutzen, ihre Verpflichtungen ebenfalls erfüllen. Damit sollte insbesondere deutlich werden, wie sehr der Erfolg eines Projekts von der verantwortungsvollen Kooperation selbständig agierender Teilgruppen abhängt.

#### **Phase 4a: Vererbung**

Mit der vierten Phase war der eigentliche Entwurf des Flugverkehrsverwaltungssystem abgeschlossen worden, da die Vererbung, welche in der folgenden Zeit in der Vorlesung besprochen wurde, bei der speziellen Projektaufgabe nur in geringem Maße zum Tragen kam. Allerdings wurde das Konzept der Vererbung in den nicht an das Projekt gebundenen Übungen an kleineren Beispielen ausführlich geübt und darüber hinaus einige denkbare Erweiterungen der Aufgabenstellung besprochen, bei denen Vererbung durchaus sinnvoll genutzt werden könnte.

Eine weitere Möglichkeit wäre gewesen, in der Vorlesung das Konzept der *deferred-Classes* nicht erst im Zusammenhang mit Vererbung sondern vorzeitig einzuführen und die gesamte Spezifikation des FEVS Systems von Anfang an durch *deferred-Classes* beschreiben zu lassen. Dies hätte erlaubt, die Implementierung

konsequent über Erbenklassen der Spezifikationsklassen realisieren zu lassen. Diese eigentlich naheliegende Idee kam uns allerdings erst während der Veranstaltung.

#### **Phase 5: Revision des Entwurfs**

Zum Abschluß des Themas "Entwurf" wurden in der Vorlesung Kriterien für einen guten Entwurf von Softwaresystemen diskutiert und an einem komplexen Beispiel ausführlich illustriert. Hierdurch wurde nochmals die Bedeutung eines übersichtlich gestalteten objektorientierten Entwurfs hervorgehoben und die Stärken und Schwächen bestimmter Vorgehensweisen besprochen. Außerdem wurde herausgestellt, daß bestimmte gute Eigenschaften eines Softwareprodukts oft Nachteile auf einer anderen Ebene mit sich bringen und eine eindeutig optimale Lösung nur sehr selten existieren kann.

Um die Fähigkeit der Studenten zur Analyse und Beurteilung von Entwürfen zu trainieren, wurden diese in der folgenden Übung aufgefordert, die angesprochenen Kriterien für einen guten Programmierstil weiter auszudiskutieren und ggf. um eigene Kriterien zu erweitern. Auf dieser Basis sollten sie dann ihren eigenen Entwurf für das FEVS System rückwirkend beurteilen und gegebenenfalls einige der früher getroffenen Entscheidungen revidieren, wenn die erkannten Nachteile die Vorteile überwiegen. Durch die rückwirkende Beurteilung ihrer eigenen Arbeit sollten die Studenten auch erkennen, welchen Lernerfolg sie im Laufe der vergangenen Wochen erzielt hatten, da sich ihr eigener Entwurfsstil mittlerweile erheblich verbessert hatte.

Im Bezug auf das Projekt war das Ziel dieser Phase, aus den früheren Einzelentwürfen einen für die gesamte Übungsgruppe verbindlichen Entwurf des FEVS Systems aufzustellen, über dessen Qualitäten sich die Studenten bewußt waren, und somit eine unabhängige Implementierung der einzelnen Klassen durch die Untergruppen vorzubereiten.

Bis zu dieser Phase war das FEVS-Projekt im wesentlichen als "Trockenübung" geplant, bei der sich die Studenten auf die Konzeption des Systems und die Beurteilung verschiedener Entwürfe konzentrieren sollten (was üblicherweise bei Programmieraufgaben erheblich zu kurz kommt) anstatt vorzeitig mit Experimenten am Computer zu beginnen. Eine stärkere Integration dieser Projektphasen in das Praktikum war zwar im Prinzip angedacht, konnte aber nicht realisiert werden, da die hierzu nötigen praktischen Vorarbeiten (vgl. Abschnitt 4) nicht rechtzeitig begonnen werden konnten.

#### **Phase 6: Implementierung und Verifikation einzelner Routinen**

Entsprechend der Grundkonzeption der Veranstaltung wurden die konventionellen algorithmischen Konzepte der Programmierung erst im letzten Drittel der Vorlesung behandelt. Der Schwerpunkt lag hierbei auf Al-

gorithmenstrukturen, Verifikation von Routinen und der Methodik des Algorithmenentwurfs, da die konkreten Ausdrücke moderner Programmiersprachen nahezu selbsterklärend sind und eine ausführliche Besprechung für Studenten eher langweilig ist.

Nachdem die Implementierung und Verifikation von Algorithmen zunächst an einigen Standardbeispielen wie dem Sortieren von Feldern geübt worden war, wurden die Studenten aufgefordert, die einzelnen von ihnen spezifizierten Klassen des FEVS Systems zu implementieren und auszutesten. Dabei war daran gedacht, daß unterschiedliche Teilgruppen einer Übungsgruppe verschiedene Klassen realisieren, so daß insgesamt eine Implementierung des Gesamtsystems entstehen konnte.

Da die Spezifikation schon relativ detailliert und mehrfach durchdacht worden war und die algorithmischen Probleme eines Flugverkehrsverwaltungssystems eher gering sind, war diese Phase konzeptuell der leichteste Teil des Projekts, erforderte aber eine Menge von Detailarbeit. Die Studenten sollten hierbei vor allem lernen, bei der Implementierung sorgfältig vorzugehen und einfache Routinen zu verifizieren, da keine Teilgruppe das gesamte System alleine erstellen konnte und sich die jeweils anderen Teilgruppen auf die Korrektheit der bereitgestellten Klassen verlassen können mußten.

Um die Studenten durch diese Aufgabe nicht zu überfordern, war daran gedacht, eine Implementierung der wichtigsten Komponenten des FEVS auf den Praktikumsrechnern bereitzustellen, so daß die einzelnen Teilgruppen jederzeit mit einem funktionsfähigen Gesamtsystem experimentieren konnten und nicht noch zusätzlich ihre eigene Testumgebung aufbauen mußten.

Bei der Planung der Leitübung wurde großer Wert darauf gelegt, naheliegende Methodiken für die Entwicklung des Systementwurfs zu verwenden, die das intuitive Verständnis der Studenten ansprechen. Auf den Einsatz der aus dem Software Engineering bekannten objekt-orientierten Analyse- und Designmethoden wurde verzichtet, da diese in den entsprechenden Hauptstudiumsveranstaltungen ausführlich behandelt werden und eine Anfängervorlesung überfrachtet hätten. Allerdings war beabsichtigt, daß die Studenten im Laufe des Projektes die Reibungsverluste einer naiven Herangehensweise bemerken und die Notwendigkeit einer strikteren Entwurfsmethodik erkennen sollten, was bei einem "perfekten" Projekt vielleicht weniger deutlich würde.

### 3 Projektübungen – Chancen und Probleme

Wir wollen in diesem Abschnitt nun eine Analyse der tatsächlich durchgeführten Projektaufgabe den Planungen des letzten Abschnitts gegenüberstellen und

die Chancen und Risiken derartiger Leitübungen in der Anfängerausbildung diskutieren. Da die Durchführung des FEVS Projekts nicht unwesentlich von den aktuellen Rahmenbedingungen der Veranstaltung "Grundzüge der Informatik I" abhing, werden wir diese zunächst kurz vorstellen, bevor wir die Ergebnisse einer lehrbegleitenden Untersuchung des Übungsbetriebs besprechen. Wir schließen unsere Analyse ab mit einer Diskussion von Erfahrungen ähnlich gelagerter Projekte.

#### 3.1 Rahmenbedingungen der Erstsemesterveranstaltung

Prägend für die Durchführung der "Grundzüge der Informatik I" ist die große Teilnehmerzahl von etwa 400 Studenten, welche organisatorische Fragen zu einem der dominierenden Faktoren der Veranstaltung werden läßt. Daher ist es nicht in jedem Zyklus möglich, die Vorlesung, die Übungen und das Praktikum von einem einzigen Veranstalter durchführen zu lassen. In diesem Fall, der auch im Wintersemester 1993/94 eintrat, sind Vorlesung und Übung eng gekoppelt, während die Details der verwendeten Programmiersprache in einer separaten "Praktikumsvorlesung" vorgestellt werden. Dies hat zum Vorteil, daß eine Trennung zwischen den grundsätzlichen Konzepten und der Besprechung von Detailfragen stattfindet, und macht es leichter, in der Vorlesung die wissenschaftlichen Aspekte der Programmierung hervorzuheben. Allerdings besteht die Gefahr einer zu starken Entkoppelung von Vorlesung und Praktikum, wenn die erforderliche sehr enge Abstimmung zwischen den Veranstaltern nicht sichergestellt<sup>2</sup> werden kann.

Im Vergleich mit dem Praktikum könnte man die Übungen als Trockenübung ansehen, innerhalb derer sich die Studenten mit konzeptionellen Aufgaben und der Beurteilung verschiedenartiger Lösungen beschäftigen sollten. Im Gegensatz dazu steht im Praktikum die selbständige Lösung von Programmierproblemen am Computer im Vordergrund. Diese unterschiedlichen Ziele haben auch unterschiedliche Organisationsformen zur Folge. Für die Bearbeitung der Praktikumsaufgaben stehen im Fachbereich mehrere Rechnerräume zur Verfügung, in denen zu bestimmten Zeiten ein Praktikurstutor als Berater bereitsteht. Die Übungen werden als (insgesamt 18) Gruppenübungen mit etwa 20–25 Teilnehmern organisiert, in denen ein Tutor die Studenten zu einer selbständigen Bearbeitung der Übungsaufgaben in Kleingruppen anlei-

<sup>2</sup>In anderen Zyklen, in denen ein Veranstalter die Kapazitäten besitzt, die gesamte Veranstaltung alleine durchzuführen, besteht diese Problematik kaum, da die Darstellung der Programmiersprache in die eigentliche Vorlesung integriert wird. Dies birgt allerdings die Gefahr einer Überbetonung der konkreten Sprache zu Lasten der wichtigen Grunddenkweisen und Konzepte in sich, so daß die Vorlesung von vielen Studenten nur als ein ausführlicher Programmiersprachenkurs angesehen wird.

ten und eine gemeinsame Besprechung der Lösungen leiten soll. Alle Übungstutoren werden durch eine von der hochschuldidaktischen Arbeitsstelle (HDA) unterstützte didaktische Schulung auf die anstehenden Aufgaben vorbereitet.

Mit Ausnahme einer Testaufgabe des Praktikums hatten alle gestellten Aufgaben einen *Angebotscharakter*. Auf einen Zwang zur regelmäßigen Mitarbeit – etwa durch Abgabe von Lösungen, wie dies in anderen Veranstaltungen des ersten Semesters üblich war – wurde verzichtet, da dieser als kontraproduktiv im Hinblick auf die Entwicklung von Selbständigkeit, Selbstdisziplin und Verantwortungsbewußtsein angesehen wurde. Schriftliche Lösungen von Aufgaben wurden daher nur auf Wunsch<sup>3</sup> korrigiert, um den Studenten ein Feedback geben zu können. Darüberhinaus wurden auf jedem Übungsblatt mehr Aufgaben gestellt, als man sinnvoll im Rahmen einer Übungsstunde bearbeiten konnte, so daß nur ein Teil der Aufgaben wirklich besprochen werden konnte, während der Rest als zusätzliches Trainingsmaterial galt. Zu allen Übungsaufgaben wurden Musterlösungen erstellt, die den Studenten in der Woche nach den Übungsstunden zur Verfügung gestellt wurden.

Wie im vorigen Abschnitt bereits angedeutet, war das FEVS-Projekt in seinen ersten Phasen im wesentlichen als Bestandteil der Übungen konzipiert, um dem Entwurf und der Beurteilung einen gebührenden Stellenwert zu verleihen. Eine engere Anbindung dieser Phasen an das Praktikum war zwar wünschenswert, konnte aber nicht rechtzeitig realisiert werden. Dies lag vor allem daran, daß die das Praktikum betreuende Rechnerbetriebsgruppe bisher nur wenig mit der objektorientierten Denkweise im allgemeinen und mit Eiffel im speziellen vertraut war und zudem in den Monaten vor der Durchführung der Lehrveranstaltung durch andere Aufgaben stark belastet war. Auch stand der Compiler für die aktuelle Version Eiffel 3.0 zu spät zur Verfügung, so daß die nötigen Vorbereitungen für die Durchführung des Praktikums erst verhältnismäßig spät abgeschlossen werden konnten. Zusätzliche Maßnahmen zur Integration des Projekts in das Praktikum konnten erst im Laufe des Semesters ergriffen werden. In den ersten Wochen des Praktikums wurden daher eher konventionelle Programmieraufgaben behandelt, während Aufgaben, die eine Implementierung des FEVS-Systems vorbereitet hätten, erst gegen Ende des Semesters gestellt wurden.

Auch die Bedingungen für eine Durchführung der Projektaufgabe im Rahmen der Übungen waren natürlich nicht in allen Punkten so optimal, wie man es sich im Interesse der Studenten und des angestrebten Lernerfolgs wünschen würde. So war auch der die Übung betreuende Assistent mit der objektorientierten Denkweise nicht so gut vertraut, um die Ausges-

<sup>3</sup> Angesichts der hohen Teilnehmerzahlen würde eine Korrektur aller Übungsaufgaben ohnehin die finanziellen Mittel des Fachbereichs überschreiten.

taltung der einzelnen Projektphasen optimal leiten zu können, und konnte wegen anderer Aufgaben erst relativ spät mit seinen Vorbereitungen beginnen. Auch gab es Unterschiede zwischen den Fähigkeiten der einzelnen Tutoren, so daß es leider nicht möglich war, für jede der 18 Übungsgruppen die gleichen Bedingungen zu schaffen. Jedoch brachten die meisten der Kleingruppen- und Praktikumstutoren sehr gute Vorkenntnisse aus den beiden vorhergehenden Zyklen und ein großes Interesse an der Vermittlung ihrer Begeisterung für Eiffel mit, so daß in Anbetracht der zusätzlichen didaktischen Schulungsmaßnahmen die Rahmenbedingungen für die einzelnen Übungsgruppen trotz der hohen Gruppenstärke als relativ gut bezeichnet werden konnten.

Dennoch zwangen uns die Rahmenbedingungen, von einer völligen Freigabe des FEVS-Projektes in die Eigenverantwortung der Studenten abzusehen, da dies eine Präzisierung der Aufgabenstellung für die einzelnen Phasen auf der Basis der Ergebnisse der vorhergehenden Phasen unmöglich gemacht hätte. Eine weniger genaue Formulierung der Aufgaben dagegen hätte von den Tutoren eine eigenständige Leitung des Projekts verlangt, was sicherlich für die meisten eine Überforderung darstellt. So wurde zu jeder Projektphase eine Musterlösung erarbeitet, auf die in den folgenden Phasen Bezug genommen wurde. Dies war zwar unbefriedigend in dem Sinne, daß die Freiheit der Studenten eingeschränkt und die Wiederverwendbarkeit ihrer Ergebnisse aus den ersten 4 Phasen begrenzt wurde, mußte aber zugunsten einer Durchführbarkeit des Projekts bei einer Gesamtteilnehmerzahl von 400 Studenten in Kauf genommen werden.

### 3.2 Analyse der Projektdurchführung

Im Rahmen einer von der HDA mitbetreuten Diplomarbeit [Hornecker, 1995] wurde lehrbegleitend eine ausführliche Analyse des Übungsbetriebs in der Informatik durchgeführt, die sich auf eine Fragebogenaktion in der Mitte des Semesters und viele detaillierte Interviews mit Studenten und Tutoren stützt.

Dabei stellte sich heraus, daß nahezu alle Studenten und Tutoren das Leitbeispiel als sehr sinnvoll beurteilten, vor allem weil es motivationsfördernd wirkte und deutlich zu Kreativität, Selbständigkeit und Teamarbeit anregte. Studenten ohne Programmierfahrung stellten fest, daß ihnen die in den Übungen stattfindende objektorientierte Modellierung der Realität leichter gefallen sei als die konventionellen Programmieraufgaben in der Anfangsphase des Praktikums. Den anderen wurden durch die Aufgabe vor allem die Vorteile der Sprache Eiffel deutlich, die sie zu Beginn meist als unnötig kompliziert (im Verhältnis zu der ihnen bekannten Sprache Pascal) angesehen hatten. Viele Studenten stellten fest, daß sie durch die Lehrveranstaltung Freude und Interesse am Studium gewonnen hätten und viel über Informatik lernen würden,

was ihrer Ansicht nach zu einem Großteil auch in der Projektübung begründet liegt. Dieser Eindruck eines größeren Lernerfolgs wurde auch objektiv durch die Ergebnisse von Semestral- und Vordiplomklausuren bestätigt.

Natürlich ist das Zusammenspiel von objektorientierten Techniken in der Anfängerausbildung und der projektartigen Leitübung als Unterstützung bei der Einführung dieser Techniken nur einer der Faktoren, die hierbei eine Rolle spielten. Dennoch scheinen die konkreten Einzelaussagen der Studenten zu bestätigen, daß dieses einer der wesentlichen Aspekte war. Aus der Umfrage ergab sich, daß der rote Faden, der sich durch Vorlesung und Übung zog, für die Studenten besonders durch das wiederholte Aufgreifen der Projektaufgabe erkennbar wurde. Dies wiederum motivierte sie zu einer beständigeren Mitarbeit, wofür auch der im Verhältnis zu den vorhergehenden Zyklen sehr geringe Rückgang der Teilnehmerzahlen in den Übungsgruppen und in der Vorlesung ein gewisses Indiz ist.

Insgesamt läßt sich also feststellen, daß der Einsatz der Projektübung in der Anfängerveranstaltung grundsätzlich positiv zu werten war. Natürlich gab es auch Kritikpunkte. Diese waren aber nicht grundsätzlichlicher Natur, sondern richteten sich eher gegen Schwachpunkte der konkreten Durchführung und sind somit als Hinweise für zukünftige Verbesserungen zu sehen.

Viele Studenten äußerten, daß ihnen die Übungsaufgaben insgesamt zu umfangreich und kompliziert erschienen seien. Dies lag zum Teil daran, daß die Übungsblätter immer mehr Aufgaben enthielten, als man in einer Übungsstunde besprechen konnte, den Studenten dieser Angebotscharakter der Übungen aber trotz expliziter Ankündigungen nicht klar genug geworden war. Zum anderen entstand dieser Eindruck sicherlich auch, weil der vorgesehene Arbeitsplan für die Leitübung nicht von Anfang an offenlag. Somit entstand eine gewisse Orientierungslosigkeit im Hinblick auf die tatsächlichen Anforderungen, die sich bei einer frühzeitigeren konkreten Vorbereitung zum Teil hätte vermeiden lassen.

Ein Dilemma liefert die Frage nach dem optimalen Freiheitsgrad der Projektdurchführung. Zum einen äußerten viele Studenten den Wunsch nach Orientierung im Sinne von Musterlösungen zu den einzelnen Übungsaufgaben, da diese ihnen ein gewisses Gefühl von Sicherheit bieten. Zum anderen bemerkten sie aber auch, daß gerade durch diese Musterlösungen ihre Freiheit bei der eigenständigen Gestaltung eines Systemkonzeptes behindert und der eigentliche Projektcharakter deutlich eingeschränkt wurde. Prinzipiell hätten die Studenten natürlich ihre eigene Lösung weiterbenutzen können, weil in es in den Übungen ja nicht um das Erstellen einer "richtigen" Lösung ging, sondern um eine erfolgreiche Bearbeitung der Problemstellung. Da sich die Formulierungen der weiteren Aufgabenstellungen jedoch zu einem gewissen Teil

an der Musterlösung orientierten, haben nur die wenigsten diese Chance zur Selbständigkeit wahrgenommen. Rückwirkend betrachtet wirkte sich die Erstellung von Musterlösungen allerdings auch kontraproduktiv auf die Teambildung aus, die nicht in allen Übungsgruppen in dem Maße stattfand, daß alle entstehenden Aufgaben wirklich auf unabhängige Teilgruppen verteilt werden können. In manchen Übungsgruppen kam deshalb nicht einmal der in Phase 5 anvisierte verbindliche eigene Entwurf für die weitere Implementierungsarbeit zustande. Anscheinend sank durch die Existenz von Musterlösungen die Notwendigkeit für eine verbindliche regelmäßige Mitarbeit der einzelnen Teammitglieder und somit auch das Gefühl für die Mitverantwortung am Gelingen des Gesamtprojekts. Da wir aus pädagogischen Gründen bewußt auf äußeren Druck im ersten Semester verzichten wollten, waren die didaktischen Fähigkeiten der Tutoren, auf eine freiwillige aktive Mitarbeit aller Gruppenmitglieder einzuwirken, in manchen Fällen sichtlich überfordert. Zugunsten einer realeren Projektgestaltung und eines höheren Lernerfolgs sollte daher auf Musterlösungen verzichtet und versucht werden, das für die Studenten notwendige Gefühl von Sicherheit auf anderen Wegen zu erreichen.

Der Freiwilligkeitscharakter von Übung und Praktikum wurde von den meisten Studenten auch im Rückblick auf den erzielten Lernerfolg als sehr positiv, wenn auch nicht ganz unproblematisch angesehen. In Anbetracht der Tatsache, daß in den anderen Pflichtveranstaltungen des ersten Semesters ein gewisser Druck zur regelmäßigen Mitarbeit ausgeübt wurde, haben viele Studenten erst sehr spät verstanden, daß auch ohne Zwang eine regelmäßigen Vor- und Nacharbeit wichtig ist. Ihre eigene Vorbereitung auf die Informatik I Übungsstunden fiel deshalb oft den Pflichten aus anderen Fächern zum Opfer. Nichtsdestotrotz hatte die Projektaufgabe eine so stark motivierende Wirkung, daß sich fast alle Studenten sehr rege an der Projektaufgabe beteiligten. Die Ergebnisse der Befragungen lassen vermuten, daß dieser motivierende Charakter der Leitübung verloren gehen würde, wenn im Interesse einer besseren Vorbereitung Druck auf die Studenten ausgeübt würde. Zugunsten eines hohen Lernerfolgs sollte daher der Freiwilligkeitscharakter erhalten bleiben und die Motivation für eine regelmäßigen Vor- und Nacharbeit auf konstruktivere Art (und durch Abbau des Drucks aus anderen Fächern) aufgebaut werden.

Ein weiteres Problem für die Studenten war die unterschiedliche Gestaltung von Übung und Praktikum, die zu Recht als ein Abstimmungsproblem bemängelt wurde. Während innerhalb von Vorlesung und Übung konsequent ein Top-Down Ansatz verfolgt wurde, bei dem die Studenten schrittweise von einem intuitiven Verständnis der Problemstellung über den Systementwurf zur Implementierung geführt wurden, begann das Praktikum eher konventionell mit der vollständigen Implementierung von "Spielbeispielen", bei denen

objekt-orientiertes Denken keinerlei Bedeutung hatte. Die sich hieraus ergebende späte Integration des Projekts in das Praktikum ließ zum Ende des Semesters die Zeit zur tatsächlichen Implementierung eines lauffähigen Gesamtsystems zu knapp werden, wodurch das Projekt für viele Studenten einen zu starken Trockenübungscharakter erhielt. Eine sehr viel engere Anbindung des Praktikums an Vorlesung und Übung, gekoppelt mit einer Neukonzeption des Praktikums, wurde deshalb von allen Beteiligten (Studenten, Tutoren und Dozenten) gewünscht, während die organisatorische Trennung der Teilveranstaltungen durchaus als Bereicherung angesehen wurde.

Die erst zum Ende des Semesters beginnenden tatsächlichen Arbeiten am Rechner führte im Endeffekt auch dazu, daß das FEVS Projekt nur von wenigen Gruppen erfolgreich bis zur Implementierung geführt wurde und viele Studenten ein praktisches Erfolgserlebnis vermissen ließ. Dies lag unter anderem daran, daß sich die Studenten während dieser letzten Projektphase bereits auf die ersten Semesterklausuren vorbereiten mußten und die Arbeit am Projekt deshalb hinten anstellten. Andererseits endeten Übung und Praktikum mit Ende der Vorlesungszeit, so daß es für die Studenten kaum noch einen Grund mehr gab, sich während der Semesterferien mit dem Projekt zu beschäftigen. Wir sehen hierin im wesentlichen ein organisatorisches Problem, das zum Beispiel durch eine zeitliche Verlagerung des Praktikums – was viele Studenten begrüßt hätten – und zusätzliche Anreize zur Mitarbeit auch in der Ferienzeit (siehe Abschnitt 4) gelöst werden könnte.

### 3.3 Erfahrungen aus ähnlichen studentischen Projekten

Erfahrungen mit studentischen Projektübungen wurden in der Literatur bisher meist im Kontext von Software Engineering Veranstaltungen berichtet. In diesen Veranstaltungen ist die Teilnehmerzahl deutlich geringer als in den Informatik-Anfängerveranstaltungen an den meisten deutschen Hochschulen. Auch sind sie von ihren Zielrichtungen her deutlich spezieller und nicht für Erstsemester ohne Vorkenntnisse gedacht. Dennoch lassen sich Ähnlichkeiten in der Gestaltung der Projektübungen erkennen und viele Anregungen für den Aufbau studentischer Projekte im Rahmen einer Lehrveranstaltung lassen sich zumindest in abgeschwächter Form übertragen.

In [Shaw & Tomayko, 1991] werden Erfahrungen mit Software Engineering Veranstaltungen im Grundstudium (undergraduate level) berichtet. Dabei wird hervorgehoben, daß eine Projektorientierung bei der Gestaltung von Programmierveranstaltungen in jedem Fall als hilfreich anzusehen ist, aber ein großes Maß von Koordination zwischen Vorlesung und Praktikum erfordert, die allerdings nur schwer zu erreichen sei. Als optimale Gruppengröße werden 15–20, höchstens

30 Personen angesehen. Diese Zahl reicht aus, um einige der Management- und Kommunikationsprobleme eines realistischen Projekts zu erzeugen, ist aber für einen einzelnen Betreuer noch handhabbar (bei studentischen Betreuern sollte man sich besser auf 15–20 Teilnehmer beschränken). Fehler zuzulassen wird als wichtiges didaktisches Mittel betrachtet, da auch die Analyse von Fehlschlägen im Projekt einen großen Lerneffekt mit sich bringt. Dementsprechend wird für die zugehörige Vorlesung eine Vermittlung der richtigen Denkweisen als besonders wichtig angesehen. Die Studenten können sich konkrete Techniken des Software Engineering, die selbstverständlich auch gelehrt werden müssen, relativ leicht aneignen, wenn sie die richtige Sicht auf das Problem entwickelt haben (während die Umkehrung keineswegs gilt).

Brügge und Coyne [Bruegge & Coyne, 1994] beschreiben Erfahrungen mit der Förderung von Teamarbeit und heben hierbei hervor, daß die Kombination von Objektorientierung und Projekt-basierten Veranstaltungen besonders gut geeignet sei, um die Studenten frühzeitig mit Problemen einer realistischen Größenordnung zu konfrontieren und ihnen die außerfachlichen Aspekte des Softwareentwicklungsprozesses nahebringen. Dabei wird empfohlen, die Lehrveranstaltung so zu dimensionieren, daß ein Projekterfolg innerhalb eines Semesters möglich ist, und dafür ggf. sogar auf einige Aspekte und Techniken des Software Engineering zu verzichten. Großer Wert wird auch darauf gelegt, den Studenten möglichst viel Freiheiten zu lassen, und die Vorlesung als eine Hilfestellung zu verstehen, Techniken zu erlernen, die für den Projektfortschritt sinnvoll sind. Auf diese Art wird die Notwendigkeit der Lehrinhalte wesentlich deutlicher und die Kreativität der Studenten freigesetzt. Daß ein Projekt trotz aller Vorbereitungsmaßnahmen scheitern kann, selbst wenn etablierte Analyse- und Designmethoden (OMT) und Kommunikationshilfsmittel (elektronisches Bulletin Board n-dim) eingesetzt werden, wird keineswegs als mangelnder Lernerfolg angesehen, zumal viele Probleme im Softwareentwicklungsprozeß nahezu unvermeidbar sind und somit den Studenten schon in ihrer Ausbildung begegnen sollten.

Sicherlich kann man die Grundtendenzen der Erkenntnisse aus Software Engineering Veranstaltungen genauso auf Anfängerveranstaltungen der Informatik übertragen, die das Thema Programmierung zum Schwerpunkt haben, zumal sie sich in großen Teilen mit unseren eigenen Erfahrungen zu decken scheinen. Was die konkrete Ausgestaltung der Projekte angeht, so wird man allerdings deutliche Abstriche gegenüber einer Spezialvorlesung (die im Hauptstudium ohnehin angeboten wird) machen müssen, um den besonderen Ansprüchen einer Einführungsveranstaltung gerecht zu werden. Hierauf wollen wir in Abschnitt 4 näher eingehen.

### 3.4 Eine zusammenfassende Beurteilung

Auch wenn die Einführung von Objektorientierung und die Integration einer projektartigen Leitübung im Prinzip zwei unabhängige Aspekte der Gestaltung einer Einführungsveranstaltung in die Programmierung sind, läßt sich feststellen, daß eine Kombination dieser beiden Aspekte mit einer entsprechend (top-down) konzipierten Vorlesung im Hinblick auf eine Steigerung des Lernerfolgs sehr erfolgversprechend ist. Hierfür gibt es eine Reihe von Gründen.

- Die Vermittlung der wissenschaftlichen Denkformen und Arbeitsweisen ist für Studenten des ersten Semesters noch wichtiger als die Darstellung von Fachwissen. Dies befähigt die Studenten, sich im Selbststudium Kenntnisse zu erarbeiten, die in Lehrveranstaltungen nicht abgehandelt werden können.
- Die objektorientierte Sicht gibt ein besseres Bild der “realen Programmierung” und macht die Bearbeitung komplexerer Aufgaben überhaupt erst möglich. Zudem ist für Studenten ohne Programmiererfahrung ein objektorientierter Entwurf leichter zu erstellen.
- Projektaufgaben unterstützen die Einführung objektorientierter Techniken, da sie ihre Vorteile deutlich machen, und fördern die Entwicklung von Kreativität, Selbständigkeit, Teamfähigkeit und Verantwortungsbewußtsein. Darüber hinaus motivieren sie zu einer regelmäßigeren Beschäftigung mit der gestellten Aufgabe.
- Eine entsprechend der Reihenfolge des Softwareentwurfs konzipierte Vorlesung macht die Natürlichkeit eines objektorientierten Programmierstils ebenso deutlich wie die Notwendigkeit von Verifikation (Zuverlässigkeit) und einer systematischen Entwicklung von Softwaresystemen. Da Konzepte vorgestellt werden, wenn sie im Projekt benötigt werden, ist ein roter Faden leichter erkennbar und die Motivation zum aktiven Mitdenken größer.
- Der Erfolg eines Projektes ist zwar wünschenswert, aber aus didaktischer Sicht nicht unbedingt erforderlich. Manchen wird die Notwendigkeit eines systematischen Vorgehens und einer zuverlässigeren Mitarbeit im Projektteam (anstelle einer “Einzelkämpferlösung”) erst richtig klar, wenn sie mit den außerfachlichen Problemen eines realistischen Softwareentwicklungsprozesses konfrontiert werden.

Da die in Abschnitt 3.2 beschriebenen Kritikpunkte nicht die eigentliche Idee der projektartigen Leitübungen berührten, sondern im wesentlichen nur mit der konkreten Durchführung zu tun hatten, lohnt es sich, das Konzept der Projektübungen weiterzuverfolgen

und im Hinblick auf eine Überwindung der angesprochenen Probleme zu verfeinern. Einige Ideen hierzu wollen wir in dem nun folgenden Abschnitt ansprechen.

## 4 Einsatz von Projektaufgaben in der Anfängerveranstaltung

Die bisherigen Erfahrungen deuten darauf hin, daß Projektaufgaben in Veranstaltungen zum Thema Programmierung ein sehr sinnvolles Mittel zur Unterstützung der Lehre darstellen, das auch in Anfängerveranstaltungen erfolgversprechend eingesetzt werden kann. Nichtsdestotrotz kann die Durchführung derartiger Projekte gerade im ersten Semester eine Reihe von Problemen mit sich bringen, die sich durch vorbereitende Maßnahmen größtenteils vermeiden lassen. Wir wollen daher in diesem Abschnitt einige Überlegungen diskutieren, die dazu beitragen können, die Chancen projektartiger Leitübungen für den Gesamtlernerfolg besser zu nutzen.

Bei allen vorbereitenden Planungen für den Einsatz von Projektaufgaben im ersten Semester ist zu berücksichtigen, daß der Einführungscharakter einer Anfängerveranstaltung im Vordergrund stehen muß. Dies bedeutet, daß mehr noch als in den Veranstaltungen der höheren Semester (siehe die Diskussion Abschnitt 3.3) die Vermittlung von wissenschaftlichen Denkformen und Arbeitsweisen wichtiger ist als eine Darstellung von Fachwissen und Techniken der Programmierung. Eine inhaltliche Überladung sollte unter allen Umständen vermieden werden, zumal die Studenten die nötigen Kenntnisse und Methoden in späteren tiefergehenden Veranstaltungen erwerben können. Sinnvoller ist dagegen eine Veranstaltung, welche die Fähigkeit der Studenten zum Anwenden von Wissen trainiert (selbst, wenn dieses Wissen noch gar nicht vollständig präsentiert wurde) und sie zu einem eigenverantwortlichen Selbststudium anregt.

Ebenso ist zu berücksichtigen, daß die Veranstaltung Erststudierende mit und ohne Programmierkenntnisse, Wiederholer, und Studenten mit Berufserfahrung oder einem abgeschlossenen Studium gleichermaßen fördern sollte. Sie muß also eine gewisse Herausforderung für gute Studenten darstellen, ohne dabei die fachlich schwächeren zu benachteiligen. Prinzipiell ist das Mittel der Projektübungen für diese Aufgabe besonders gut geeignet, da die Programmierung im Großen auch für die besseren Studenten etwas Neues darstellt, während der objektorientierte Entwurf den Studenten ohne Programmiererfahrung leichter fallen wird als ein konventioneller.

Aus diesem Grunde sollte sich die Planung der gesamten Veranstaltung an den methodischen Zielsetzungen des Projekts orientieren (und nicht umgekehrt). Die Vorlesung sollte diejenigen Kenntnisse vermitteln, die für das Projekt relevant sind, und einen Top-Down Zugang zum Thema Programmierung

favorisieren, so daß neue Konzepte immer dann zur Sprache kommen, wenn sie für die nun anstehende Projektphase erforderlich<sup>4</sup> sind.

Die Übungen sollten einerseits dazu dienen, daß die Studenten die einzelnen Konzepte zunächst isoliert voneinander anhand einfacher Problemstellungen verstehen. Andererseits sollten sie im Rahmen des Projektes ein Diskussionsforum für die Erarbeitung des Systementwurfs darstellen, wobei die entsprechenden Aufgabenblätter eine Anleitung zur Weiterbearbeitung des Projektes sein sollten. Um dieser Doppelfunktion gerecht zu werden (und Studenten, die zeitweilig den Einstieg verloren haben, den Wiedereinstieg zu erleichtern), sollte höchstens in jeder zweiten Übungsstunde auf das Projekt eingegangen werden.

Auch des Praktikum sollte sich dementsprechend an einem Top-Down Zugang orientieren, was zumindest an der TH Darmstadt eine teilweise Neukonzeption verlangt. Da das Ziel eines Praktikums ist, die Studenten Erfahrungen durch praktische Übungen am Rechner sammeln zu lassen, mußten die Studenten früher alle Bestandteile der Programmiersprache kennen, die für eine eigenständige Lösung einer Problemstellung erforderlich sind. Dies führte zwangsweise zu einem Bottom-Up Zugang und dazu, daß die Studenten im Praktikum mit kleinen Spielbeispielen beginnen mußten. Studenten ohne Programmiererfahrung sind hierbei besonders benachteiligt.

Zugunsten einer konsequenteren Integration der objektorientierten Denkweise sollten die praktischen Übungen den Aspekt der Wiederverwendbarkeit stärker betonen und die Studenten schrittweise von der Rolle eines *Benutzers existierender Klassen* in die eines *Entwicklers neuer Klassen* überführen. Als vorbereitende Maßnahme bietet sich hierzu die in [Meyer, 1993] vorgeschlagene Bereitstellung von *Black Boxes*<sup>5</sup> an. Black Boxes befreien die Studenten in der Anfangsphase des Praktikums davon, Routinen implementieren zu müssen (was sie ja noch nicht gelernt haben), und gibt ihnen dennoch die Möglichkeit, Programme zu schreiben, die "etwas tun". Eine schrittweise Öffnung dieser Black Boxes ermöglicht ihnen später dann, Implementierungen zu analysieren, zu beurteilen und somit implizit als Vorbilder für die Entwicklung eigener Programme zu verwenden. Die eigene aktive Programmierfähigkeit im Sinne einer Implementierung komplizierterer Algorithmen kann somit auf einen Zeitpunkt verschoben werden, zu dem in der Vorlesung über algorithmische Konzepte und Kriterien für die Entwick-

<sup>4</sup>So sollten z.B. zu Beginn nur die Konzepte von Klassen und Systemstrukturen besprochen werden und von den *algorithmischen* Ausdrücken nur diejenigen angesprochen werden, die zur Benutzung existierender Klassen (durch Routinenaufruf, Zuweisung etc.) erforderlich sind.

<sup>5</sup>Da im Gegensatz zum Wintersemester 1993/94 gute Eiffel-Compiler mittlerweile sogar für PC's erhältlich sind, könnten zur Entlastung der üblicherweise überlaufenen Rechnerpools der Universitäten diese Black Boxes den Studenten auch für Experimente am eigenen Rechner bereitgestellt werden.

lung guter Programme bereits ausführlich gesprochen wurde, ohne daß es bei reinen Trockenübungen bleiben muß.

Für eine stärkere Integration des Projekts in das Praktikum wäre zusätzlich der Einsatz aufgeschobener Klassen (*deferred Classes*) sinnvoll, die dementsprechend frühzeitig im Rahmen der Vorlesung besprochen werden sollten – also weit vor dem Konzept der Vererbung, in das sie normalerweise eingebettet sind. Dies würde den Studenten ermöglichen, ihren *Systementwurf* frühzeitig im Rahmen des Praktikums zu implementieren und das Zusammenspiel der Klassen auch praktisch zu überprüfen, während die Ausprogrammierung einzelner Routinen auf später verschoben werden kann. Somit kann auch in den frühen Phasen des Projekts ein Trockenübungscharakter vermieden werden, ohne daß die eigentliche Entwurfsphase zu kurz kommt.

Für die konkrete Gestaltung des Projekts wäre eine Gesamtgruppengröße von 15–20 Studenten ideal (an den meisten Universitäten aber wohl aus finanziellen Gründen nicht realisierbar). Auf die betreuenden Tutoren käme vor allem die Aufgabe zu, den Studenten ein reales Projektgefühl zu vermitteln, die Bildung von kleineren Teams zu fördern, diese bei der Bearbeitung ihrer Problemstellungen zu beraten und die Kommunikation zwischen den einzelnen Teams der Gesamtgruppe zu unterstützen. Da dies von den Tutoren viel Einfühlungsvermögen und Flexibilität, ein hohes Maß von Motivation und ein tiefes Verständnis der Materie verlangt, ist es wichtig, sie durch eine entsprechende didaktische Schulung auf ihre Aufgabe vorzubereiten und sie auch während der Durchführung der Lehrveranstaltung intensiv zu betreuen. Unsere bisherigen Erfahrungen mit der derartigen didaktischen Maßnahmen waren ausgesprochen positiv.

Während die Entwurfsphase des Projekts im Verlaufe der Vorlesungszeit abgeschlossen werden kann, scheint es empfehlenswert, die Implementierungsphase bis zu einer festgesetzten Abgabefrist innerhalb der ersten Wochen der Semesterferien auszudehnen. Dies würde eine vom Veranstaltungsrhythmus unabhängige Projektarbeit ermöglichen, welche von den Studenten selbständig organisiert werden könnte und somit einen echten Projektcharakter hätte. Zudem würde die wichtige und arbeitsintensive Endphase des Projekts nicht durch Vorbereitungen auf Semestralklausuren und ähnliche Prüfungen gestört, einem vielfach geäußerten Wunsch der Studenten auf sinnvolle Weise Rechnung getragen und die Chance auf einen Projekterfolg deutlich gesteigert. Die zeitliche Belastung der Studenten innerhalb der Vorlesungszeit müßte natürlich dementsprechend verringert werden.

Nicht unkritisch ist die Frage nach dem Grad der Autonomie, die man den Studenten bei der Bearbeitung ihres Projektes im Rahmen von Übung und Praktikum einräumen sollte. Der von uns bevorzugte Verzicht auf regelmäßigen Kontrollen läßt im Prinzip auch

zu, daß sich einzelne Studenten überhaupt nicht an den Übungen beteiligen und somit der Projekterfolg einer ganzen Gruppe gefährdet wird. Ob diesem Problem allerdings durch Druck zur regelmäßigen Mitarbeit, etwa durch abzugebende Übungsaufgaben oder Programmierstate, erfolgreich begegnet werden kann, erscheint uns zumindest zweifelhaft. Ein derartiger Zwang erzeugt meistens das Gegenteil von dem, was angestrebt wird: die meisten Studenten unterlaufen den Druck durch Abschreiben von Lösungen oder Kopieren von Programmen und sehen die Notwendigkeit zur aktiven Mitarbeit um so weniger ein, da sie den Nutzen nicht erkennen können.

Unserer Ansicht nach ist Freiwilligkeit gekoppelt mit dem Versuch, zur Mitarbeit zu überzeugen, der sinnvollere Weg. Selbständigkeit, Kreativität, Teamfähigkeit und Verantwortungsbewußtsein kann nur reifen, wenn Studienanfänger eigenständige Entscheidungen auch über ihre Form der Mitarbeit fällen dürfen, selbst wenn sie dabei Gefahr laufen, schlechte Entscheidungen zu treffen. Zudem entspricht es (auch nach [Bruegge & Coyne, 1994]) dem Projektcharakter besser, nur das endgültige Ziel vorzugeben, und die konkrete Arbeitsform einem sozialen Aushandlungsprozeß innerhalb der einzelnen Großgruppe zu überlassen. Der hierbei entstehende, durch das gemeinsam zu erreichende Ziel motivierte soziale Druck auf die einzelnen Gruppenmitglieder paßt eher zu der Realität eines Projekts als eine ständige Überprüfung durch den Veranstalter.

Natürlich kann man nicht ganz auf die Vorgabe von Spielregeln verzichten, da ein gewisses Maß an Orientierung für die Studenten unumgänglich ist. Diese sollten sich aber auf Zielvorgaben (Aufgabe und Projektende), Bewertungskriterien und ähnliche Rahmenbedingungen beschränken und ergänzt werden durch Anreize zur einer aktiven regelmäßigen Mitarbeit sowie Vorschläge, wie dies sinnvoll geschehen könnte. Ein möglicher Anreiz wäre zum Beispiel das Angebot, den (in Darstadt erforderlichen) Übungsschein wahlweise durch eine Semestralklausur oder ein erfolgreich abgeschlossenes (Teil-)Projekt zu erlangen und die Note durch den gewählten Schwierigkeitsgrad festzulegen. Ein entsprechender Vorstoß im Wintersemester 1995/96 stieß bei den Studenten durchaus auf ein positives Echo, muß aber noch ausgewertet werden.

Für die meisten Studienanfänger wird ein Projekt eine völlig neuartige und ungewohnte Lernform darstellen. Von großer Bedeutung für ihren Lernerfolg ist daher, einem Gefühl der Orientierungslosigkeit und Überforderung vorzubeugen, indem die an sie gestellten Anforderungen, die Rahmenbedingungen und Freiheitsgrade sowie die vorgesehene Vorgehensweise mit Entwurfs-, Erprobungs- und Implementierungsphasen den Studenten von Anfang an offenliegen (was natürlich eine langfristige Vorbereitung von Vorlesung, Übung und Praktikum voraussetzt). Hierbei sollte insbesondere auch die gegenseitige Abhängigkeit der Mit-

glieder eines Projektteams hervorgehoben werden und den Studenten deutlich gemacht werden, daß sie in Übungen und Praktikum von Beginn an eine aktive Rolle übernehmen sollten, um durch eigene Erfahrungen gleichzeitig fachliche Fertigkeiten als auch notwendige soziale Fähigkeiten wie Teamarbeit und Verantwortungsgefühl zu erlernen.

Letztendlich hat auch die Art der gewählten Aufgabenstellung einen gewissen Einfluß auf den Projekterfolg. Als Leitübung zur Förderung eines objektorientierten Programmierstils bieten sich dabei besonders *Simulationsaufgaben* in der Art des vorgestellten Flugverkehrsverwaltungsystems an, bei denen ein naher Bezug zur Realität besteht und die zu programmierenden Probleme intuitiv leicht verständlich sind. Bei einer entsprechenden Unterstützung im Praktikum (die praktische Handhabbarkeit sollte ohnehin vorher überprüft worden sein) sind somit die praktischen Auswirkungen des eigenen Entwurfs bzw. der eigenen Implementierungen für die Studenten jederzeit erkennbar.

## 5 Ausblick

Der Einsatz projektartiger Leitaufgaben in der Informatik-Grundlehre bietet vielfältige Chancen für eine Steigerung des studentischen Lernerfolgs und eine Reihe von Vorteilen bei der Vermittlung von wissenschaftlichen Denkformen, Konzepten und Methoden der Programmierung. Die Studenten können frühzeitig an Probleme einer realistischen Größenordnung herangeführt werden, erkennen hierdurch die Vorteile der objektorientierten Vorgehensweise, entwickeln ein größeres Interesse für eine eigene aktive Mitarbeit in der Lehrveranstaltung und werden vor allem zu Kreativität, Teamarbeit und Selbständigkeit angeregt.

Die Integration derartiger Projekte in eine Anfängerveranstaltung erfordert allerdings deutlich aufwendigere Vorbereitungsmaßnahmen als dies bei konventionellen Übungen der Fall ist. Vorlesung, Übung und Praktikum müssen projektorientiert konzipiert und sehr eng aufeinander abgestimmt werden. Die einzelnen Projektphasen der Aufgabe selbst müssen sehr gut vorbereitet und auf ihre praktische Realisierbarkeit überprüft worden sein. Das größere Ausmaß an Freiheiten bei der Bearbeitung einer Projektaufgabe erfordert fachkundige und gut geschulte Tutoren, die in der Lage sind, die Bildung von echten Arbeitsteams zu fördern. Es muß dem Eindruck begegnet werden, daß ein systematischer objektorientierter Zugang zur Programmierung etwas Unpraktisches sei, der zu Beginn besonders unter Studenten mit Programmiererfahrungen in konventionellen Programmiersprachen entstehen kann.

Mit den in dieser Arbeit dargestellten Überlegungen und Erfahrungen des "Darmstädter Modells" wollten wir eine Anregung geben, wie Anfängerveranstaltungen zum Thema Programmierung in einer Art gestal-

tet werden können, die für alle Beteiligten interessant und erfolgreich werden kann. Natürlich stellten die im vorhergehenden Abschnitt diskutierten Vorschläge keine vollständige und allgemeingültige Liste von Empfehlungen dar. Vielmehr sollten sie ständig erweitert und an lokale Gegebenheiten angepaßt werden. Da jedoch ein Großteil unserer Vorschläge bei der Gestaltung der Anfängerveranstaltung im Wintersemester 1995/96 aufgegriffen und erfolgreich angewandt werden konnte, sind wir überzeugt, daß sie dazu beitragen können, den Einsatz von Projektaufgaben im ersten Semester zu einem empfehlenswerten Lehrkonzept für das Informatik-Grundstudium werden zu lassen.

## Literatur

- [Bruegge & Coyne, 1994] Bernd Brügge and Robert F. Coyne. Teaching iterative and collaborative design: lessons and directions. In J. L. Diaz-Herrera, editor, *Software Engineering Education, 7th SEI CSEE Conference*, LNCS 750, pages 411–427. Springer Verlag, 1994.
- [Gries, 1981] David Gries. *The science of programming*. Springer Verlag, 1981.
- [Henhagl & et. al, 1994] W. Henhagl and et. al. Erfahrungsbericht über zwei Jahre “Grundzüge der Informatik I / II”. FB Informatik, TH Darmstadt, 1994.
- [Hornecker, 1995] Eva Hornecker. Grundzüge der Informatik I – didaktische Analyse des Übungsbetriebs. Diplomarbeit, TH Darmstadt, März 1995.
- [Kreitz, 1994] Christoph Kreitz. Grundlagen der Informatik I: Programmierung. Skriptum zur Vorlesung, TH Darmstadt, 1994.
- [Meyer, 1988] Bertrand Meyer. *Object-oriented Software Construction*. Prentice Hall, 1988.
- [Meyer, 1992] Bertrand Meyer. *Eiffel – the Language*. Prentice Hall, 1992.
- [Meyer, 1993] Bertrand Meyer. Toward an object-oriented curriculum. *Journal of object-oriented programming*, 1993.
- [Shaw & Tomayko, 1991] Mary Shaw and James E. Tomayko. Models for undergraduate project courses in software engineering. In James E. Tomayko, editor, *Software Engineering Education, 5th SEI Conference*, LNCS 538, pages 33–65. Springer Verlag, 1991.