# Verwendung objektorientierter Konzepte im Software-Praktikum

Doris Schmedding

Fachbereich Informatik
Lehrstuhl für Software-Technologie
Universität Dormund

**Zusammenfassung.** In einem Praktikum am Ende des Grundstudiums, in dem die Programmierausbildung in einer objektorientierten Sprache erfolgt, werden Studentengruppen komplexe realitätsnahe Aufgaben gestellt. Die Analyse der Lösungen zeigt, dass objektorientierte Konzepte beherrscht werden. Durch eine Befragung der Studierenden wird überprüft, ob ihre eigene Einschätzung mit den Beobachtungen übereinstimmt.

## 1 Einleitung

Seit einigen Jahren erfolgt die Grundausbildung im Informatikstudium an der Universität Dortmund in einer objektorientierten Programmiersprache, zunächst in BETA, seit drei Jahren in Java [DoDi99]. Die Lehrveranstaltungen „Programmierung“ im ersten Semester und „Algorithmen und Datenstrukturen“ im dritten Semester sind klassisch aufgebaut: In der Vorlesung werden neue Konzepte vorgestellt, die in darauf ausgerichteten Übungsaufgaben von den Studierenden gezielt angewendet werden. Im Software-Praktikum am Ende des Grundstudiums setzen die Studierenden die erworbenen Programmierkenntnisse ein, um realitätsnahe Aufgaben zu lösen.

In der Programmierungsvorlesung [Gru99] wird über den Begriff des Objekts, objektorientierte Analyse und objektorientierten Entwurf an die konkrete Ausprägung Java herangeführt. Da bei der Konzeption dieser Lehrveranstaltung davon ausgegangen werden muss, dass die Studierenden keine Programmiervorkenntnisse besitzen, nimmt die Vermittlung allgemeiner (nicht objektorientierter) Sprachkonstrukte einen großen Raum ein. Obwohl es auch Gründe dafür gibt, in der Ausbildung erst strukturierte Programmierung und danach objektorientierte Konzepte zu behandeln [ABB99], wurde in der Programmierausbildung mit Java mit objektorientierten Konzepten begonnen, um so das objektorientierte Denken besser zu schulen.

Bei der Korrektur der Lösungen der Praktikumsaufgaben fällt auf, dass die einzelnen Praktikumsgruppen die objektorientierten Sprachkonstrukte in unterschiedlichem Maße nutzen. Manche Gruppen setzen sie sehr souverän ein, andere scheinen sie mehr zu meiden. Wir wollen die Frage untersuchen, ob die Konzepte durch die Übungen zu den Grundvorlesungen ausreichend verinnerlicht sind. Der Unterschied zwischen dem Verstehen eines Sprachkonzepts und dem Beherrschen ist beträchtlich, die Studierenden müssen selbst erkennen, wie das neue objektorientierte Sprachkonstrukt sinnvoll zur Lösung ihres Problems eingesetzt werden kann. Wenn sich die

Aufgabe auch mit anderen, stärker präsenten Sprachkonstrukten lösen lässt, scheint die Vermeidung der neuen Sprachkonstrukte sinnvoll und effizient, um Fehler zu vermeiden, da am Ende des Praktikum eine funktionsfähige Lösung erwartet wird. Nur wenn die neuen Konzepte von den Studierenden als vorteilhaft eingesehen werden, lassen sie sich auch in den Lösungen nachweisen.

Nach der Vorstellung der Lehrveranstaltung Software-Praktikum mit ihrem Vorgehensmodell werden eine Praktikumsaufgabe und die Lösungen der Studententeams beschrieben. Die Praktikumsteilnehmer wurden mittels Fragebogen zu ihrer Einstellung zur Objektorientierung befragt. Einige Ergebnisse der Befragung werden vorgestellt und den Beobachtungen bei der Analyse der Lösungen gegenübergestellt.

## 2 Das Software-Praktikum

Das Software-Praktikum (SoPra), das einen Umfang von 5 SWS hat, ist eine Pflichtlehrveranstaltung für alle Studierenden der Informatik in Dortmund. Gruppen von 8 Studierenden führen unter Anleitung eines Betreuers nacheinander zwei Projekte durch. Zur Zeit nehmen 90 Studierende aufgeteilt in 12 Gruppen am SoPra teil. Voraussetzung für die Teilnahme ist für KerninformatikerInnen die bestandene Vordiplomsprüfung über die Programmierungslehrveranstaltung. Für die Studierenden der angewandten Informatik, kurz Ingenieurinformatiker genannt, gibt es keine Zulassungsbeschränkungen.

Das Praktikum zielt neben der Vertiefung der Programmierkenntnisse darauf ab, handlungsorientiert Methoden und Verfahren aus der Software-Technik zu vermitteln. In den frühen Entwicklungsphasen wird zur Modellierung die Unified Modeling Language (UML) [HiKa99] und das Werkzeug TogetherJ eingesetzt. Die Softwareentwicklung folgt einem vorgegebenen einfachen **Vorgehensmodell** [Schm01], das an das Vorgehensmodell von Hitz und Kappel [HiKa99] angelehnt ist.

In der Anforderungsdefinitionsphase wird ein Anwendungsfalldiagramm und ein Klassendiagramm erstellt, das den Problembereich beschreibt. Anwendungsfälle werden durch Aktivitätsdiagramme näher beschrieben. Das Problembereichsmodell stellt ein reines Datenmodell dar, die Klassen enthalten nur Attribute, noch keine Methoden. Mit Hilfe eines graphischen GUI-Builders wird ausgehend von den Anwendungsfälle die graphische Benutzungsschnittstelle des Programms gestaltet, wobei die Swing-Bibliothek benutzt wird. Das zu entwickelnde Programm entspricht dem Model-View-Control-Konzept. In der Analysephase werden die Klassen des Problembereichsmodells in das Strukturmodell übernommen. Zur Realisierung der Anwendungsfälle werden Control-Klassen hinzugefügt. Mit Hilfe von Sequenzdiagrammen wird untersucht, welche Methoden die Klassen zur Realisierung der Anwendungsfälle benötigen. Das programmiersprachenunabhängige Modell wird in der Entwurfsphase an die Implementierungssprache Java angepasst. Die Typisierung von Attributen und Parametern erfolgt. Es wird festgelegt, welche Bibliotheksklassen benutzt werden. Die Integration von GUI-Klassen und Modellklassen wird entworfen. wobei die Control-Klassen die Schnittstelle zwischen GUI und Modell bilden. In der Implementierungsphase erfolgt die Implementierung der Klassen, die vor ihrer Integration unabhängig voneinander mit Hilfe von Dummy-Klassen getestet werden. Da das Werkzeug TogetherJ Java-Code-Rahmen generiert, besteht die Implementierungsarbeit im Wesent-

lichen aus der Codierung der Methoden-Bodies und der Ereignisbehandlung in den GUI-Klassen. Den Abschluss bildet der Produkttest.

## 3 Aufgabenstellung und Lösungen

Typische Aufgabenstellungen stammen aus der Verwaltung (Autovermietung, Bibliothek usw.). Alle 12 Praktikumsgruppen bearbeiten die gleiche Aufgabe, die Verteilung von Übungs-, Seminar- und Projektgruppenplätzen. Studenten melden sich zu diesen Lehrveranstaltungen an, können bei Projektgruppen und Übungen Prioritäten für Themen bzw. Termine und Freundesgruppen angeben, mit denen sie zusammenarbeiten wollen. Die Verteilung der Plätze erfolgt für die drei Lehrveranstaltungstypen unterschiedlich. Lehrende richten Lehrveranstaltungen ein und nehmen an einigen Stellen Einfluss auf die Verteilung, wenn sie z.B. Projektgruppenteilnehmer nach fachlichen Gesichtspunkten auswählen.

Alle Lösungen zeigen, dass die Studierenden in der Lage sind, objektorientierte Modelle zu erstellen. Die Lösungen zeichnen sich durch eine konsequente Verwendung von Kapselung und Abschottung aus, was sich zum einen auf die Ausbildung in der Programmierungsvorlesung, aber auch auf den Entwicklungsprozess und den von den Werkzeugen generierten Code zurückführen lässt. Unser Hauptaugenmerk liegt auf dem Modell der Applikationsklassen ohne die GUI-Klassen. Die Basis für den Vergleich der Lösungen liefern die erstellten UML-Diagramme.

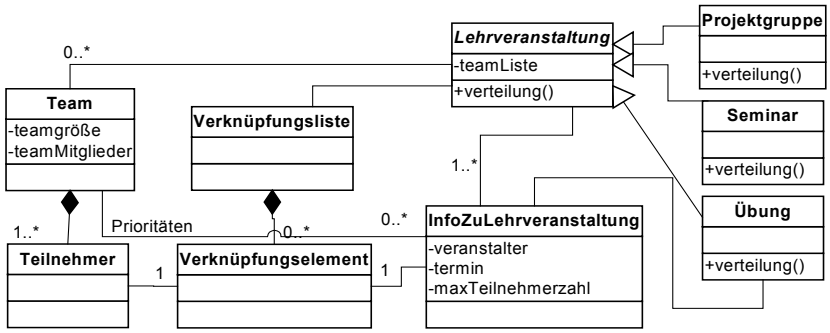



**Abb. 1:** Verteilung von Projektgruppen, Seminar- und Übungsplätzen (Ausschnitt)

Die Lösungen unterscheiden sich deutlich in der Anzahl der modellierten Klassen (9 bis 30) und in der Verwendung objektorientierten Konzepte. Aspekte des Problems, wie z.B. der Veranstalter einer Lehrveranstaltung, werden als Klasse (vgl. Abb.2) oder als Attribut (vgl. Abb.1) modelliert. Abb.1 zeigt eine der weniger umfangreichen Lösungen, die sparsam objektorientierte Konzepte (eine abstrakte Klasse mit einer abstrakten Methode) einsetzt. Nur eine Gruppe verzichtete ganz auf Vererbung in ihren selbst entworfenen Klassen, weil die Ausprägungen der Lehrveranstaltungen zu wenig ähnlich seien und die Verteilungsverfahren sich stark unterscheiden würden. Vererbungsbeziehungen beschränken sich oft auf die Erweiterung einer Oberklasse um Attribute und Methoden.

Listenstrukturen werden von allen Gruppen als Spezialisierung von `LinkedList` oder `Vector` realisiert. Teilweise sind dazu eigene erbende Klassen

(`Verknüpfungsliste` in Abb.1) mit eigener Schnittstelle entworfen worden, teilweise sind die Listen als Attribute (`teamMitglieder` in `Team`) mit nur der ererbten Schnittstelle entworfen worden. Polymorphie findet sich in fast allen Modellen in der Form von Listen bestehend aus Objekten verschiedener Unterklassen (eine Veranstaltungsliste enthält Seminare, Projektgruppen und Übungen). Wenn eine abstrakte Klasse wie `Lehrveranstaltung` in Abb.1 definiert wird, enthält sie oft auch eine abstrakte Methode wie `verteilung()`. Die „echte" Redefinition, das Überschreiben einer bereits implementierten Methode, findet sich nicht.

Manche Lösungen (vgl. Abb.2) zeigen einen geradezu verschwenderischen Umgang mit Vererbung. Es werden zwei abstrakte Klassen, Person und Veranstaltung, eingesetzt. Die Verteilung wird hier mit Hilfe eines Interfaces realisiert, das von den verschiedenen Verteilverfahren implementiert wird. In dieser Lösung sieht man noch deutlich die Strukturen des Problembereichsmodells durchschimmern.

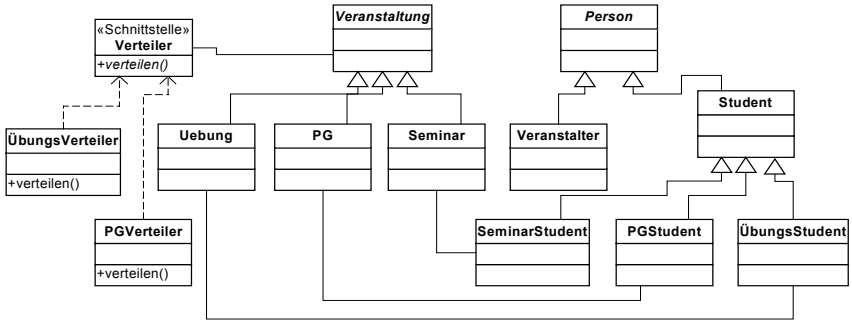



**Abb. 2:** Ausschnitt aus einer anderen Studentenlösung

Da die graphische Benutzungsschnittstelle unter Verwendung von Swing mit Unterstützung des graphischen GUI-Builders Forté realisiert wird, werden hier objektorientierte Konzepte konsequent eingesetzt. Selbst einfachste Dialoge lassen sich ohne Vererbung nicht realisieren, denn sie stellen Spezialisierungen der Klasse `JFrame` dar. Auch die Umsetzung des Event-Mechanismus verlangt das Verständnis der Objektorientierung. Werden Listen oder Tabellen in der GUI dargestellt, was für Verwaltungsaufgaben typisch ist, muss man sich intensiv mit dem Model-View-Control-Konzept auseinandersetzen. Swing stellt Klassen (wie z.B. `DefaultListModel` und `ListSelectionModel`), abstrakte Klassen (`AbstractListModel, AbstractTableModel`) und Interfaces (`ListModel, ListDataListener`) für unterschiedliche Vorgehensweisen bei der Realisierungen zur Verfügung.

Alle erstellten Programme zeichneten sich durch sehr ansprechend gestaltete graphische Oberflächen aus. Auch die funktionalen Anforderungen wurden von fast allen Teams erfüllt. Nur eine Gruppe konnte kein funktionstüchtiges Programm erstellen. Drei Programme zeigten einige Einschränkungen im Funktionsumfang.

## 4 Befragung der Studierenden

Mit Hilfe eines Fragebogens wurden die 90 SoPra-TeilnehmerInnen zur Verwendung von objektorientierten Konzepten befragt. Fast alle Teilnehmer sind im 4. Semester, 3 studieren Lehramt Informatik, 40 Kerninformatik und 47 Ingenieurinformatik.

75 Fragebögen wurden ausgefüllt. 85 Prozent der Befragten geben an, vor Java eine andere Programmiersprache gelernt zu haben, dabei werden Pascal und Basic etwa gleich häufig genannt. Die Vermutung, dass eine imperativ geprägte Vorbildung vom Gebrauch objektorientierter Konzepte abhält, fand keine Bestätigung. 80 Prozent geben an, heute überwiegend objektorientiert zu programmieren. Nur 4 Teilnehmer beurteilen ihre eigenen Programmierfähigkeiten als sehr gut, 19 beurteilen sie als gut. Etwa die Hälfte behauptet von sich, das zu beherrschen, was benötigt wird. 8 beurteilen ihre Fähigkeiten als „mäßig“. Als schlecht schätzt sich niemand ein.

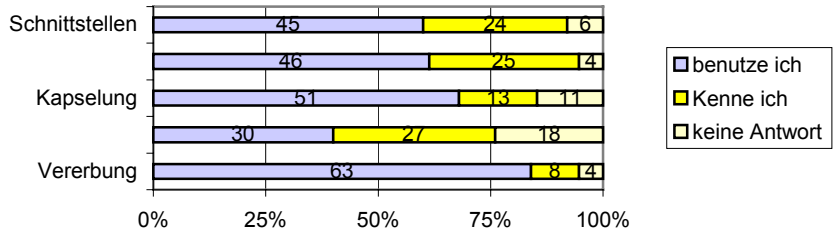



**Abb. 3:** Verwendung objektorientierter Konzepte

Ein Block von Fragen untersucht, wie weit die in der Programmierungsvorlesung eingeführten objektorientierte Konzepte(Vererbung, Polymorphie und dynamisches Binden, Kapselung und Abschottung, abstrakte Klassen und Schnittstellen) bekannt sind und verwendet werden (vgl. Abb.3). Die Antworten bestätigen die Ergebnisse der Analyse der Modelle, dass die Konzepte **Kapselung** und **Vererbung** beherrscht werden. Nur wenige (4) geben dazu keine Antwort. Nur wenige Studenten (8 bzw. 13) geben an, diese Konzept zwar zu kennen, aber nicht zu benutzen. Eine derartige Antwort lässt auf ein tiefes Unverständnis schließen. Auch **abstrakte Klassen** und **Interfaces** werden noch von 46 bzw. 45 Teilnehmern benutzt. Aber bereits ein Drittel der Befragten gibt an, dieses Konzept zwar zu kennen, aber nicht zu benutzen. **Polymorphie** scheint für die Studierenden besonders schwierig zu sein. Hier werden die wenigsten Antworten gegeben und noch nicht einmal die Hälfte der Befragten benutzt nach eigenem Bekunden Polymorphie.

Mit gezielten Fragen, welche Konzepte wann im Projektablauf eingesetzt werden, wird überprüft, ob die eigene Einschätzung dem tatsächlichen Gebrauch entspricht. Auch hier zeigt die große Masse der Antworten, dass das Erlernte souverän eingesetzt wird: Es werden Kapselung und Vererbung genannt, es wird auf die Bibliotheksklassen und die graphische Benutzungsschnittstelle verwiesen. Aus einigen wenigen Antworten (von Studierenden, die noch keine Prüfung über den Programmierungsstoff abgelegt haben) wird deutlich, dass nicht erkannt wurde, dass in allen Entwicklungsphasen objektorientierte Konzepte eingesetzt werden.

Danach folgen einige Meinungsäußerungen zur Objektorientierung, denen die Befragten zustimmen oder die sie ablehnen können. Mehr als die Hälfte ist der

Ansicht, dass sich auch ohne Objektorientierung gut strukturierte Programme schreiben lassen. Einige schränken diese Aussage auf kleine Programme ein. Viele wollen hierzu keine Stellung beziehen. Sehr deutlich (58) ist die Zustimmung zur Behauptung, dass Programme durch Objektorientierung eine klarere Struktur bekommen. Unsicherheit im Gebrauch von objektorientierten Konzepten wollen sich nur 8 Studierende eingestehen, 7 wollen lieber auf Polymorphie verzichten, um Fehler zu vermeiden.

## 5 Zusammenfassung und Bewertung

Im Software-Praktikum können wir den souveränen Einsatz der Objektorientierung, die Beherrschung der neu erlernten Konzepte beobachten. Die Klassen aus den Java-Bibliotheken werden an die eigenen Anforderungen angepasst, um graphische Benutzungsschnittstellen zu gestalten und komplexe Datenstrukturen aufzubauen.

Da es in einem studentischen Praktikum in der Regel keine Wiederverwendung von selbst entwickelten Klassen gibt, besteht für studentische Software-Entwickler keinen äußeren Grund, allgemeingültige Klassen zu entwerfen, die durch Spezialisierung an die aktuelle Aufgabenstellung angepasst werden. Deshalb finden sich in erster Linie Lösungen, die genau auf das gegebene Problem zugeschnitten sind. Aus dem gleichen Grund finden wir auch keine Redefinition, außer der Redefinition von abstrakten Methoden.

In einem Praktikum besteht das Problem, dass die Teams aus Studierenden mit heterogenem Kenntnisstand zusammengesetzt sind. Die Lösungen werden gemeinsam erarbeitet und verbergen die Unwissenheit einzelner, die sich aus manchen Fragebögen ablesen lässt. Die vorgestellten empirischen Ergebnisse belegen aber, dass die große Mehrheit die in der Grundausbildung vermittelten objektorientierten Konzepte kreativ und sinnvoll zur Lösung der Aufgaben einsetzt.

Die Meinungsäußerungen zeigen, dass Objektorientierung als sinnvoll angesehen wird und sich die Studierenden in ihrem Gebrauch sicher fühlen. Durch ein Praktikum am Ende des Grundstudiums erwerben die Studierenden Erfahrungswissen in der Verwendung objektorientierten Konzepte und vertiefen ihre Kenntnisse.

## Literatur


DoDi99 E.-E. Doberkat, S. Dißmann: Einführung in die objektorientierte Programmierung mit JAVA, Oldenbourg, 1999.

Gru99 V. Gruhn: Folien zur Vorlesung “Programmierung”, Dortmund, WS 1999/2000, (http://ls10.www.cs.uni-dortmund.de/)

HiKa99 M. Hitz, G. Kappel: UML@Work - Von der Analyse zur Realisierung, dpunkt.verlag, 1999.

Schm01 D. Schmedding: Ein Prozessmodell für das Software-Praktikum, in: H. Lichter, M. Glinz (Hrsg.) SEUH 7 – Software Engineering im Unterricht der Hochschulen, Zürich, dpunkt.verlag, 2001.

ABB99 E. Angster, J. Bergin, L. Böszörményi: Introducing OO Design and Programming with Special Emphasis on Concrete Examples, in: A. Moreira, S. Demeyer (Eds.): Object-Oriented Technology: ECOOP’99 Workshop Reader, Lisbon, Portugal, 1999, Lecture Notes in Computer Science, Vol. 1743