Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

Ein Prozessmodell für das Software-Praktikum

Doris Schmedding
Fachbereich Informatik, Lehrstuhl für Software-Technologie, Universität Dortmund
Doris.Schmedding@cs.uni-dortmund.de

Zusammenfassung

Es wird ein besonders für die Lehre in Form eines Praktikums geeignetes auf UML und Java basierendes Prozessmodell vorgestellt. Dazu werden spezielle Anforderungen an ein Prozessmodell für die Lehre im Grundstudium herausgearbeitet. Während des ersten Einsatzes wurde die Eignung des Modells durch eine Evaluation überprüft und Verbesserungsvorschläge erarbeitet.

1 Einleitung

Die Umstellung der Informatik-Grundausbildung an der Universität Dortmund auf die Programmiersprache Java machte eine grundsätzliche Umgestaltung des zum Grundstudium gehörenden Software-Praktikums (SoPra) nötig. Ziel des Praktikums ist eine handlungsorientierte Einführung in Methoden und Verfahren der Software-Technik, indem die Studierenden im Team Software-Projekte durchführen. Das Software-Praktikum ist eine Pflichtveranstaltung für alle Informatikstudierenden und wird in der Regel als sechswöchige Blockveranstaltung in der vorlesungsfreien Zeit durchgeführt. Das Praktikum findet nach dem 3. Semester statt, nachdem die Programmierausbildung abgeschlossen ist.

Es werden pro Jahr drei Praktika angeboten, an denen in der Regel 96 Studierende teilnehmen, die in 12 Gruppen zu je 8 aufgeteilt werden. Die Gruppen führen nacheinander zwei Software-Projekte durch, wobei alle Teams die gleichen Aufgaben lösen. Jedes Entwicklerteam wird von einer studentischen Hilfskraft oder einem wissenschaftlichen Mitarbeiter betreut, die den Gruppen beratend zur Seite stehen.

Als Modellierungssprache wurde UML [3] [5] [9] gewählt. Die Modellierung mit UML wird durch das relativ einfach zu bedienende Modellierungswerkzeug TogetherJ unterstützt. Die Suche nach einem für die Lehre geeigneten Prozessmodell gestaltete sich recht schwierig, denn die meisten der in der Literatur vorgestellten Prozessmodelle [7] schienen wegen ihrer Komplexität nicht geeignet zu sein. Dies gilt insbesondere für den speziell auf UML abgestimmten Rational Unified Process (RUP) [6]. Aufgrund der speziellen Anforderungen in der Lehre, die im folgenden Kapitel näher erläutert werden, wurde ein besonders einfaches SoPra-Prozessmodell

Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

entwickelt, dessen besondere Eignung für die Lehre durch eine Evaluation überprüft wurde.

Die Evaluation wurde im Rahmen einer Diplomarbeit [1] praktikumsbegleitend durchgeführt. Dazu wurden einerseits die TeilnehmerInnen mittels Fragebögen und Stundenzettel über Lernerfolg, Zufriedenheit und Arbeitsbelastung befragt, andererseits auch ihre Modellierungsergebnisse untersucht. Da 12 Gruppen parallel die gleichen Aufgaben gelöst haben, ergab sich die einmalige Gelegenheit, die Ergebnisse verschiedener Gruppen zu vergleichen.

2 Anforderungen an das Prozessmodell

Um zu einem geeigneten Prozessmodell zu gelangen, muss zunächst die Zielsetzung des Praktikums herausgestellt werden. Folgende Lernziele bilden die Grundlage des Konzepts der Lehrveranstaltung:

- Kennen lernen von Projektarbeit,
- Arbeiten im Team,
- Anwendung von Software-Entwicklungsmethoden und
- Einsatz von Software-Entwicklungswerkzeugen.

Das eingesetzte Prozessmodell muss möglichst optimal auf die beteiligten Personen, ihre Fähigkeiten, ihr Vorwissen und ihre Motivation, auf die eingesetzten Methoden und Verfahren und auf die zu verwendenden Entwicklungsumgebung mit ihren Werkzeugen abgestimmt sein [8]. Nur dann können mit diesem Entwicklungsprozess vorhersagbare Resultate erzielt werden.

Bei der Definition des Prozessmodells muss deshalb zunächst das Vorwissen der EntwicklerInnen berücksichtigt werden. Die TeilnehmerInnen am Software-Praktikum bringen nur recht rudimentäre Kenntnisse der UML mit, die sie im Rahmen ihrer Programmierausbildung erworben haben. Eine fundierte Ausbildung in Software-Technik erfolgt in Dortmund erst im Hauptstudium.

Zum Praktikum gehört eine Vorlesung im Umfang von nur einer Semesterwochenstunde, in der das Prozessmodell vorgestellt werden muss, die Sprachkonstrukte der UML und Themen aus der Programmierung wie z.B. die graphischen Benutzungsschnittstellen vertieft werden müssen, die in der Programmierungsvorlesung nur am Rande behandelt werden. Ganz neue Themen wie das Testen kommen hinzu.

Daraus ergeben sich folgende Anforderungen an ein Prozessmodells für das Software-Praktikum:

- Es muss auch ohne Vorwissen leicht erlernbar sein.
- Es muss klar strukturiert und damit leicht überschaubar sein, damit die Studierenden immer wissen, was als Nächstes zu tun ist.
- Es muss viele Hilfsangebote für die Studierenden beinhalten.

- Das Prozessmodell soll die studentischen EntwicklerInnen unterstützen, sie dürfen sich aber nicht gegängelt fühlen. Sie sollen das Prozessmodell als hilfreich und sinnvoll erleben.
- Das Prozessmodell muss auf die begrenzten zeitlichen Ressourcen des Software-Praktikums Rücksicht nehmen.

Die Komplexität der UML als Sammlung vieler Notationen bereitet große Probleme beim Einsatz in einer Lehrveranstaltung. Welche Notationen sollen in welcher Tiefe gelernt werden? Selbst wenn Zeit genug vorhanden wäre, alle Diagrammtypen und Sprachkonstrukte komplett vorzustellen, braucht es viel Erfahrung in der Modellierung, um entscheiden zu können, welcher Diagrammtyp bei der Lösung welcher Detailfrage besonders hilfreich ist. Also müssen die Lehrenden eine Teilmenge der UML festlegen, die ihrer Meinung nach groß genug ist, die Aufgaben zu lösen, und die Kreativität der EntwicklerInnen nicht einschränkt, andererseits klein genug ist, um von den Lernenden nach kurzer Zeit beherrscht werden zu können.

Auch auf den begrenzten zeitlichen Rahmen des Software-Praktikums muss bei der Definition des Prozessmodells Rücksicht genommen werden. Für ein Projekt stehen in einem Blockpraktikum nur drei Wochen zur Verfügung. Diese Zeit sollte optimal ausgenutzt werden, so dass kein Leerlauf entsteht, also alle Teilnehmer in allen Entwicklungsphasen Aufgaben haben und möglichst keine Entwickler auf Ergebnisse anderer Entwickler warten müssen.

Daneben sollte ein Prozessmodell immer möglichst gut auf die damit zu erstellenden Produkte zugeschnitten sein. In einer Lehrveranstaltung bietet sich im Gegensatz zur Industrie, wo zur vorgegebenen Aufgabenstellung das optimale Prozessmodell gesucht wird, die Möglichkeit, durch geschickte Wahl der Aufgabenstellung Prozessmodell und Aufgabe in Einklang zu bringen. Wir haben deshalb zunächst auf die Erstellung verteilter Software-System verzichtet und konnten so auch auf die Verteilungsdiagramme der UML verzichten.

Bei der Festlegung der Rollen im Prozessmodell werden in der Lehre andere Ziele verfolgt als bei der kommerziellen Software-Entwicklung, wo sich alle Entscheidungen am wichtigsten Ziel Wirtschaftlichkeit orientieren müssen. In der Lehre und besonders in der Grundausbildung sollte bei der Verteilung der Teilaufgaben an die Gruppenmitglieder nicht das effiziente Spezialistentum im Vordergrund stehen, sondern alle Lernenden sollten möglichst weitgehend mit allen im Prozessmodell vorgesehenen Aktivitäten Erfahrungen sammeln.

Auch bei Werkzeugauswahl und Einsatz müssen die besonderen Anforderungen einer Lehrveranstaltung berücksichtigt werden. Auch hier steht die leichte Erlernbarkeit und einfache Bedienung im Vordergrund der Auswahlkriterien.

3 Das SoPra-Prozessmodell

Angelehnt an das ISP-Modell (*Irrational Separated Process*) von Hitz und Kappel [5], das seinen Namen zur Abgrenzung gegenüber dem *Rational Unified Process*

Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

(RUP) [6] trägt, wurde unter Berücksichtigung der oben definierten Anforderungen das Sopra-Prozessmodell definiert. Um den Lernenden die Orientierung zu erleichtern, wurde im Gegensatz zum RUP die Abfolge der Aktivitäten recht starr festgelegt, die Aktivitäten wurden genau bestimmten Phasen zugeordnet. Erfahrene Entwickler würden sich durch ein derart starres Schema unnötig gegängelt fühlen.

Abbildung 1 zeigt den Phasenablauf des Prozessmodells, dessen Grobstruktur weitgehend vom ISP-Modell übernommen wurde. Die Anforderung, dass leichte Erlernbarkeit und Überschaubarkeit für einen in der Lehre einsetzbaren Prozess wesentliche Merkmale sein müssen, rechtfertigt seine an das Wasserfallmodell angelehnte Struktur. Am Ende jeder Phase präsentieren die Studierenden im Rahmen eines Reviews ihre Ergebnisse. Außerdem werden die von den Studierenden erstellten Dokumente von den Gruppenbetreuern korrigiert. So soll sichergestellt werden, dass nur nach vollständiger Bearbeitung einer Phase in die nächste gewechselt wird

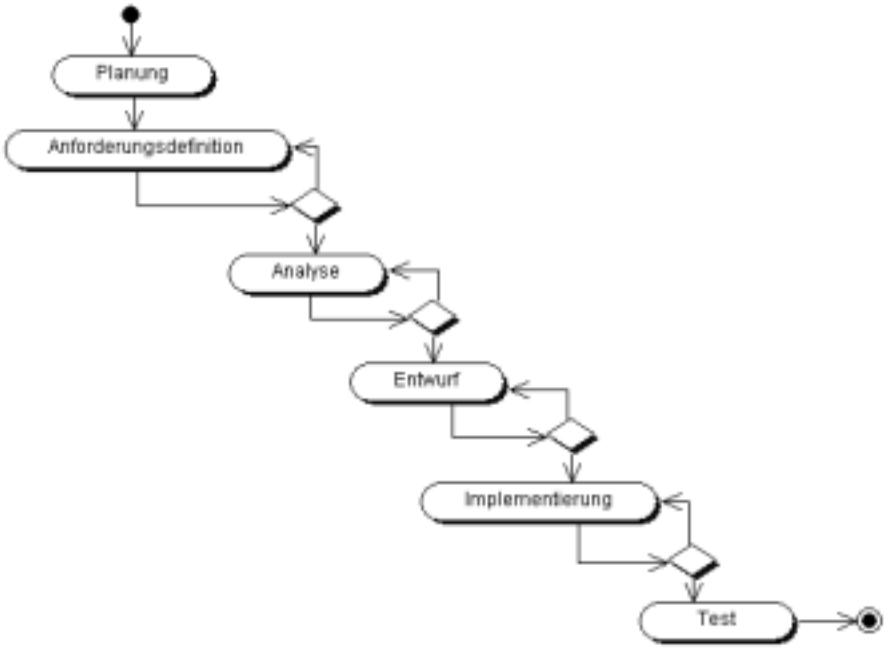

Abb. 1: Einteilung des Prozesses in Phasen

Normalerweise enthält ein Prozessmodell wie z.B. auch das ISP-Modell vage Anweisungen beispielsweise derart, dass besonders komplexe Anwendungsfälle durch Aktivitätsdiagramme näher zu untersuchen sind oder dass die Zustände von Objekten mit nicht trivialem Lebenszyklus durch Zustandsübergangsdiagramme modelliert werden sollten. Entscheidungen, wo sich welcher Aufwand lohnt, um zu neuen Erkenntnissen zu gelangen, können aber Lernende im Grundstudium mangels

Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

Erfahrung noch nicht treffen. Deshalb werden diese Arbeitsaufträge durch Anweisungen derart ersetzt, dass z.B. für alle Anwendungsfälle Aktivitätsdiagramme zu erstellen sind. So erhalten alle Gruppenmitglieder die Chance, sich an der Erstellung der Aktivitätsdiagramme zu beteiligen und diese Fertigkeit zu erlernen.

Die **Planung** eines Projekts, die neben der Aufgabenbeschreibung auch die Erstellung eines Zeitplans beinhaltet, ist allein Aufgabe der Praktikumsveranstalter. Die Studierenden wären zumindest im ersten Projekt mit der Erstellung eines Zeitplans überfordert.

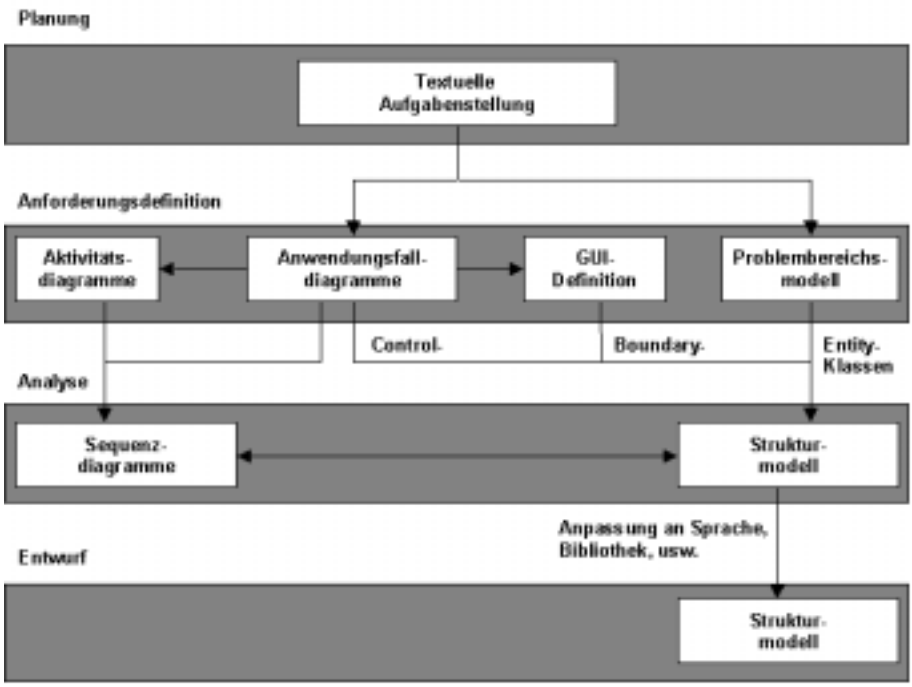

Abb. 2: Abhängigkeiten zwischen den in den Phasen eingesetzten UML-Diagrammen

Abbildung 2 zeigt, welche UML-Diagramme ausgehend von der textuellen Aufgabenbeschreibung in welchen Phasen zu entwickeln sind und welche inhaltlichen Zusammenhänge zwischen den Diagrammen bestehen. Nachfolgend wird die Zielsetzung der einzelnen Entwicklungsphasen und die darin zu erledigenden Aktivitäten näher erläutert.

Die Ausgangsbasis für die **Anforderungsmodellierung** bildet das aus den textuellen Aufgabenstellung zu erarbeitende Anwendungsfalldiagramm. Die einzelnen Anwendungsfälle werden durch Aktivitätsdiagramme näher beschrieben. Ein erstes Klassendiagramm stellt das Datenmodell des Problembereichs dar, das auch auf den Informationen des Aufgabentextes basiert. Mit Hilfe eines GUI-Builders wird ausgehend von den Anwendungsfällen die graphische Benutzungsschnittstelle des Produkts gestaltet. Die Benutzungsschnittstelle und die Funktionalität des Produkts wird

im Benutzungshandbuch beschrieben. Während in der Anforderungsdefinitionsphase alle im RUP- und auch im ISP-Prozess vorgesehenen UML-Diagramme erstellt werden, wurde in den nachfolgenden Phasen auf einige Diagramme verzichtet.

In der **Analysephase** werden die Klassen des Problembereichsmodells als sogenannte Entity-Klassen in das Strukturmodell, das Klassendiagramm der Analyse, übernommen. Zur Realisierung der Anwendungsfälle werden Control-Klassen in das Strukturmodell aufgenommen, wobei pro Anwendungsfall eine Methode vorzusehen ist. Zusätzlich kommen aus der graphischen Gestaltung der Benutzungsschnittstelle die GUI-Klassen als Boundary-Klassen hinzu.

Zu jedem Anwendungsfall, der durch ein Aktivitätsdiagramm näher beschrieben wird, wird ein Sequenzdiagramm erstellt. Mit Hilfe der Sequenzdigramme wird untersucht, welche Methoden benötigt werden. Da Sequenz- und Kollaborationsdiagramme beide die Interaktion der Objekte darstellen, wurde zugunsten der übersichtlicheren Sequenzdiagramme auf die Kollaborationsdiagramme verzichtet. Die Sequenzdiagramme wurden bevorzugt, weil sie wegen der vorgegebenen Struktur für Ungeübte leichter zu erstellen und zu verstehen sind, horizontal werden die am Anwendungsfall beteiligten Objekte dargestellt und die vertikale Dimension entspricht der Zeitachse. Der einzige Vorteil der Kollaborationsdiagramme wäre gewesen, dass sie neben den Interaktionen auch die aus dem Strukturdiagramm übernommen Objektbeziehungen darstellen. Werkzeuge sind in der Lage, diese beiden Diagrammtypen ineinander zu überführen.

Auch auf die Erstellung von Zustandsübergangsdiagrammen, mit denen Übergange zwischen den Zuständen eines Objekts modelliert werden können, wurde verzichtet. Der ISP-Prozess von Hitz und Kappel [5] sieht sie in der Analysephase vor. Zustandsübergangsdiagramme machen zwar für bestimmte komplexe Objekte Sinn, die einen nicht trivialen Lebenszyklus aufweisen, - dies gilt z.B. für manche Klassen der GUI und die Systemschnittstellenklassen - aber der Aufwand für die Vermittlung und Einübung einer weiteren graphischen Notation hätte meiner Meinung nach in keinem Verhältnis zum didaktischen Gewinn gestanden.

In der **Entwurfsphase** wird das Strukturmodell zur Vorbereitung der Implementierung an die eingesetzte Programmiersprache und die verwendeten Bibliotheksklassen angepasst. Dazu wird der Detailentwurf der Klassen vorgenommen, im wesentlichen werden dabei die Parameter der Methoden und die Sichtbarkeit der Attribute und Methoden festgelegt. In der Regel kommen im Entwurf weitere Boundary-Klassen hinzu, um z.B. die Persistenz der Daten zu realisieren. Die Entwickler können in interessanten Fällen noch einmal Sequenzdiagramme einsetzen, deren Notation bereits bekannt ist, um das Zusammenwirken der neu hinzugekommenen mit den alten Klassen zu untersuchen. Verlangt werden Sequenzdiagramme im Entwurf aber nicht.

Zu den weiteren Aufgaben der Entwurfsphase gehört die Strukturierung und Partitionierung des Strukturmodells. Falls das Modell unüberschaubar geworden ist, sollte es in Pakete aufgeteilt werden.

Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

Für das **Implementieren** und **Testen** wurde die Regel festgelegt, dass alle Gruppenmitglieder mindestens eine Klasse selbständig implementieren und nach einem vorgegebenen Verfahren testen müssen. Auf die Erstellung von Komponenten- und Verteilungsdiagrammen wurde verzichtet, sie eher der Software-Dokumentation dienen. Ihr Nutzen wird erst in der im Praktikum fehlenden Wartungsphase sichtbar, so dass sie in einer Lehrveranstaltung wie dieser schlecht zu motivieren sind.

4 Die Evaluation des Prozessmodells und einige Ergebnisse

Um den Reifegrad des vor dem ersten Software-Praktikum mit Java und UML definierten Prozessmodells gemäß Capability Maturity Model (CMM) [8] zu erhöhen, wurde eine Evaluation des Prozessmodells durchgeführt. So wurden Verbesserungsvorschläge erarbeitet, die teilweise bereits umgesetzt wurden, bzw. noch umgesetzt werden sollen. Obwohl die PraktikumsteilnehmerInnen im Software-Praktikum zum ersten Mal Software mit Unterstützung eines Prozessmodells entwickelten, befindet sich der Prozess nicht auf der initialen Reifestufe, denn die langjährige Erfahrung der Veranstalter, die den Prozess definiert haben und überwachen, verhindert chaotischen Verhaltensweisen, die typisch für diesen Entwicklungsstand sind.

Gemäß [10] lässt sich die durchgeführte Evaluation folgendermaßen klassifizieren: Evaluationsobjekt ist das neu definierte Prozessmodell des Software-Praktikums. Die Evaluation wurde praktikumsbegleitend im Februar und März 2000 in summativer Form durchgeführt. Ziel der Evaluation war herauszufinden, wie weit die in Kap. 2 definierten Anforderungen vom definierten Prozessmodell realisiert werden. Die Evaluation wurde mittels Stundenzetteln, Fragebögen und Messungen der erstellten UML-Modelle mit Hilfe einer für diesen Zweck entwickelten Metrik [1] durchgeführt.

In der industriellen Software-Entwicklung spiegelt sich die Qualität des Prozesses in erster Linie in der Qualität der erstellten Produkte wieder. Da aber das Ziel eines Praktikums nicht das Produkt, sondern der Lernfortschritt der Studierenden ist, muss nicht nur die Qualität des Endprodukts, sondern auch die Qualität der Zwischenprodukte, der während des Entwicklungsprozesses erstellten UML-Diagramme und anderen Dokumente, bewertet werden. Einerseits werden die erstellten Diagramme und Dokumente von den Praktikumsbetreuern korrigiert, andererseits wurde versucht, die Qualität der Diagramme durch die Definition einer Metrik messbar zu machen.

Angelehnt an das Goal-Question-Metric-Paradigma [2] wurden im Rahmen einer Diplomarbeit [1] die drei Maße visuelle Größe, informative Größe und Komplexität definiert, wobei mit der visuellen Größe die Überschaubarkeit der Diagramme und mit der informativen Größe der Informationsgehalt eingeschätzt werden soll. Zur Beurteilung der Komplexität der verschiedenen Diagrammtypen wurden jeweils eigene Bewertungskriterien entwickelt. Neben dieser Bewertung der erstellten Modell von außen mit Hilfe der Metrik und durch die Betreuer wurden die Studierenden am

Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

Ende des Praktikums zu einer eigenen Einschätzung ihres Lernerfolgs aufgefordert. Nachfolgend werden einige Ergebnisse vorgestellt:

Erhebliche Produktivitätssteigerung. Wöchentlich wurde in beiden Projekten durch Stundenzettel die Arbeitsbelastung der Gruppenmitglieder abgefragt. Im 2. Projekt wurde für die Modellierung nur halb so viel Zeit wie im erste Projekt benötigt, obwohl die Aufgabenstellung im 2. Projekt umfangreicher war. Die Steigerung der Produktivität im 2. Projekt ohne den Einarbeitungsaufwand in die neuen Methoden und Werkzeuge wurde von den Veranstaltern erheblich unterschätzt.

Lernfortschritte. Mittels Fragebögen wurde der Kenntnisstand, der Lernfortschritt und die Zufriedenheit der Studierenden mit dem Prozessmodell, mit den Methoden und den eingesetzten Werkzeugen ermittelt. Die Studierenden bewerteten selbst ihre Kenntnisse im Bereich der Software-Entwicklung, der Prozessmodelle, der UML, in der Programmierung allgemein und in der Programmierung in Java vor und nach dem Praktikum mit Schulnoten. In allen Bereichen wurden die Kenntnisse nachher als verbessert beurteilt, den größten Zuwachs erzielten die Gebiete Software-Entwicklung, Prozessmodelle und UML mit einer Verbesserung von 0,85, 1,3 bzw. 1,4 Punkten. Dieses Ergebnis entspricht genau der Zielsetzung des Praktikums, das Software-Technikinhalte und nicht Programmieren lehren will.

Prozessmodell. Für die Studierenden war das definierte Prozessmodell trotz einführender Vorlesung nur schwer durchschaubar. Im ersten Projekt gab ein Drittel der Teilnehmer an, nicht immer gewusst zu haben, was als nächstes zu tun sei. Insbesondere die von ihnen erwarteten Ergebnisse und die einzusetzenden Notationsmittel waren ihnen unklar. Selbst im 2. Projekte herrschte hier noch viel Ratlosigkeit.

Aktivitätsdiagramme. Neben Fragen mit vorbereiteten Antworten enthielt der am Ende des Praktikums ausgeteilte Fragebogen auch offen formulierte Fragen, nach Eindrücken und Meinungen zum Praktikum. Die Antworten der TeilnehmerInnen machten deutlich, dass das Erstellen der Aktivitätsdiagramme als besonders lästig und wenig nutzbringend empfunden wurde. Einerseits scheint die Bedeutung dieses Diagrammtyps für den gesamten Entwicklungsprozess nicht richtig deutlich geworden zu sein, andererseits wurden weisungsgemäß sehr viele, auch winzig kleine Aktivitätsdiagramme erstellt. Insgesamt wurden 245 Aktivitätsdiagramme entwickelt, pro Gruppe und Projekt 10 bis 15 Diagramme.

Sequenzdiagramme. Die Modellierung der Sequenzdiagramme wurde als besonders aufwendig und schwierig empfunden. In der ersten Version waren sie meist unvollständig oder enthielten Fehler. Die mangelnde Qualität spiegelte sich auch in den gemessenen Größen wieder, die informative Größe und die Komplexität der zunächst erstellten Diagramme waren gering. Deshalb mussten sehr viele Sequenzdiagramme überarbeitet werden. Die Studierenden gaben an, für die Modellierung der Sequenzdiagramme etwa gleich viel Zeit wie für die Klassendiagramme benötigt zu haben, obwohl an den Klassendiagrammen in allen Phasen gearbeitet wurde, während die Sequenzdiagramme nur in der Analysephase eingesetzt wurden.

Anwendungsfalldiagramme. Die Anwendungsfalldiagramme haben sich als Einstiegspunkt in die Modellierung bewährt. Sie wurden als nützlich und wenig auf-

wendig empfunden, ihre Modellierung kostete die wenigste Zeit von allen Diagrammtypen.

Klassendiagramme. Der Nutzen der Modellierung der Klassendiagramme wurde von allen Teilnehmern eingesehen. Unterstützt wurde dieser positive Eindruck sicher vom eingesetzten Modellierungswerkzeug TogetherJ, da zu jeder Klasse schon während ihrer Modellierung ihr Java-Code angezeigt wird. Die gemessenen Größen zeigten an, dass der Informationsgehalt der Klassendiagramme von Phase zu Phase wuchs.

Warum wurden manche Diagrammtypen abgelehnt?

Die Evaluation zeigte einige Stärken und Schwächen der vorgestellten Vorgehensweise auf. Die von den Studierenden geäußerte Unzufriedenheit machte sich im Wesentlichen an den Aktivitätsdiagrammen und an den Sequenzdiagrammen fest, während Klassendiagramm und Anwendungsfalldiagramm als nützlich und den Entwicklungsprozess unterstützend akzeptiert wurden.

Aktivitätsdiagramme bereiten den Studierenden Schwierigkeiten, da der in der Vorlesung vorgestellte Sprachumfang nur eine geringe Teilmenge der vom Tool angebotenen und in Lehrbüchern [5] behandelten Ausdrucksmöglichkeiten darstellt. Während wir Aktivitätsdiagramme recht informal in der Anforderungsdefinitionsphase zur Beschreibung von Anwendungsfällen einsetzen, können sie auch in der Analyse und im Detailentwurf zum Entwurf von Prozeduren einsetzt werden [6]. Entsprechend mächtig ist die Notation, die zur Beschreibung der Abläufe verwendet werden kann, beispielsweise sieht die Notation grafische Elemente für das Senden und Empfangen von Ereignissen, Objekte und Objektfluss vor. Dieser Widerspruch zwischen der gelehrten und der vom Tool angebotenen Notation scheint für die Verunsicherung der Studierenden mitverantwortlich zu sein.

Hinzu kommt, dass die Festlegung einer geeigneten Granularität der modellierten Aktivitäten für Ungeübte schwierig ist. Wird zu grob modelliert, sind die Diagramme wenig hilfreich beim nächsten Modellierungsschritt. Wird die Granularität der Aktivitäten zu fein gewählt, ist die Erstellung der Diagramme sehr aufwendig. Die Studierenden haben zwar sehr viele aber sehr kleine Diagramme modelliert. Um den Aufwand für diesen Diagrammtyp insgesamt für sie akzeptabel zu halten, haben sie, da sie alle Anwendungsfälle durch Aktivitätsdiagramme beschreiben sollten, sehr grob modelliert. Da die Aktivitätsdiagramme zu einem sehr frühen Zeitpunkt in unserem Entwicklungsprozeß erstellt werden, sind den Studierenden vielleicht die Abläufe noch nicht klar genug, um tiefer vordringen zu können.

Sequenzdiagramme dienen dazu, die Dynamik des Systems zu untersuchen. Die Betrachtung der dynamischen Abläufe wird von den Studierenden grundsätzlich als schwieriger empfunden als die Untersuchung statischer Strukturen. In einem Sequenzdiagramm wird beschrieben, wie ein Methodenaufruf bei einem Objekt weitere Methodenaufrufe bei anderen Objekten hervorruft, die wiederum Methodenaufrufe bei weitern Objekten bewirken. Bei der Korrektur der Diagramme fiel auf, dass bei

der Modellierung nicht weit genug in die Tiefe des Systems vorgedrungen wurde. Außerdem bereitet den Studierenden die Darstellung von Iterationen Probleme.

Sowohl aus den Aktivitätsdiagrammen als auch aus den Sequenzdiagrammen wird kein Code erzeugt, so dass ihr direkter Nutzen für die Produktentwicklung nicht offensichtlich ist.

5 Konsequenzen aus den Ergebnissen

Da das Praktikum handlungsorientiert angelegt ist, besteht die Gefahr, dass die Studierenden die Zielsetzung missverstehen. Das Ziel ist nicht ein Software-Produkt, sondern das Erlernen des systematischen Vorgehens bei der Software-Entwicklung. Wenn das allen Teilnehmern klar ist, sollte auch die Motivation der UML-Diagramme, die nicht direkt zu Java-Code führen, leichter fallen.

Die berechtigte Forderung nach mehr Information über den Ablauf des Prozessmodells führte zu einer detaillierten Beschreibung des Modells auf den WWW-Seiten des Software-Praktikums (http://ls10-www.informatik.uni-dortmund.de/LS10/Pages/sopra-specials/Phasenmodell.html). Beispiele zu den einzelnen Diagrammtypen zeigen, wie die Notation eingesetzt wird und bis zu welchem Detaillierungsgrad modelliert werden soll. Da in der Vorlesung aus Zeitgründen und aus Gründen der Überschaubarkeit nur kleine Beispiele vorgestellt werden können, wirkt der Einsatz bestimmter Diagrammtypen dort übertrieben, denn die Zusammenhänge lassen sich auch ohne diese Diagramme leicht durchschauen. Komplexe Beispiele zum Selbststudium sind deshalb zur Motivation wichtig.

Von den Studierenden kann verlangt werden, dass sie sich selbst besser auf die Aufgaben im Praktikum vorbereiten. Um eine intensivere Nachbereitung der Vorlesungsinhalte anzuregen, wurden Präsensaufgaben zu Beginn der einzelnen Entwicklungsphasen eingeführt: In Multiple-Choice-Tests wird der Vorlesungsstoff abgefragt und die Lösungen in der Gruppe besprochen.

Die Gruppenbetreuer sollen die Gruppen in Fragen des Methoden- und Werkzeugeinsatzes beraten. Wenn unter den Studierenden große Unsicherheit herrschte über den Sinn bestimmter Diagramme und die einzusetzenden Notationsmittel, müssen auch die Gruppenbetreuer fachlich besser auf ihre Aufgaben vorbereitet werden. Ihnen wurde deshalb zur Aufgabe gestellt, die oben beschriebenen Multiple-Choice-Tests auszuarbeiten.

Auf jeden Fall sollen auch weiterhin im Praktikums zwei Projekte durchgeführt werden, denn so erleben die Studierenden den Produktivitätsgewinn und die Entlastung durch Erfahrung. Erst im zweiten Projekt können sie die neu erlernten Notationen wirklich kreativ zur Beschreibung ihrer Vorstellungen einsetzen.

Als Konsequenz aus dem überraschend hohen Produktivitätsgewinn wurde im nachfolgenden Praktikum der Umfang der ersten Aufgabe auf ein Minimum reduziert und die Komplexität der zweiten Aufgabe erhöht. Man könnte auch neue Inhalte in das 2. Projekt verlagern, etwa die Entwicklung eines verteilten Systems als Aufgabe stellen.

Die erstmals für die Evaluation des Prozessmodells eingesetzten Stundenzettel wurden in den nachfolgenden Praktika beibehalten. Sie können den Veranstaltern und den Teilnehmern wichtige Informationen über den Schwierigkeitsgrad der Aufgabenstellungen, über den Zeitplan des Projekts, die Gleichmäßigkeit der Aufgabenverteilung in der Gruppe usw. liefern.

Obwohl die aus didaktischen Gründen vorgenomme Einschränkung der UML mitverantwortlich ist für einige der aufgezeigten Probleme, sehe ich keine Alternative zu diesem Vorgehen. Aus Zeitgründen kann nicht die vollständige UML gelehrt werden. Die Studierenden wären auch nicht in der Lage, alles bei ihrer Modellierung einzusetzen. Der Verzicht auf die wenig geliebten Diagrammtypen kann keine Lösung sein, vielmehr muss versucht werden, ihren Nutzen für den Entwicklungsprozess deutlicher hervorzuheben.

Insgesamt bestätigen die Ergebnisse der Evaluation die Richtigkeit der Strategie, die UML weitgehend einzuschränken und ein möglichst einfaches Vorgehensmodell für das Praktikum auszuwählen.

Danksagung. Ganz herzlich bedanken möchte ich mich bei Heiko van Elsuwe, der im Rahmen seiner Diplomarbeit die Evaluation durchgeführt hat, die einige Schwachpunkte des definierten Prozessmodells offengelegt und Verbesserungsmöglichkeiten aufgezeigt hat.

Literatur

1. Heiko van Elsuwe: Optimierung eines Prozessmodells durch Evaluation, Diplomarbeit am Fachbereich Informatik, Universität Dortmund, 2000.
2. Norman E. Fenton, Shari Lawrence Pfleeger: Software Metrics - A Rigorous & Practical Approach. International Thomson Computer Press, 1996
3. Martin Fowler: UML Distilled - Applying the Standard Object Modelling Language. Addison-Wesley, 1997.
4. Watts S. Humphrey: A Discipline for Software Engineering. Addison- Wesley, 1995.
5. Martin Hitz, Gerti Kappel: UML@Work - Von der Analyse zur Realisierung. dpunkt-Verlag, 1999.
6. Phillipe Kruchten: The Rational Unified Process: An Introduction. Addison-Wesley, 1998.
7. Gunter Müller-Ettrich: Objektorientierte Prozessmodelle. Addison-Wesley, 1999.
8. Mark C. Paulk, Charles V. Weber, Bill Curtis, Mary Beth Chrissis: The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1995.
9. James Rumbaugh, Ivar Jacobson, Grady Booch: The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.
10. Heinrich Wottawa, Heike Thierau: Evaluation. Verlag Hans Huber, 1990.