Einsatz von objektorientierten Technologien und Softwareentwicklungsprozessen in der Lehre

Stefan Traub

Abstract: Der Einsatz objektorientierter Technologien ist seit längerer Zeit Standard in der Lehre. Oftmals fehlt es aber an deren sinnvoller Umsetzung im Rahmen von Projekten. Aus diesem Grunde wird an der Fachhochschule Ulm seit drei Semestern im Fach "Allgemeine Informatik" (Studienarbeiten) die Nutzung von OO-Technologien praktiziert. Die Studenten arbeiten in Gruppen in Form von Projektteams zusammen. Bei der Erstellung der Software ist ein Softwareentwicklungsprozess obligatorisch. Die Erfahrung nach einigen Semestern zeigt, dass dadurch die Qualität der Software deutlich gesteigert werden konnte und auch die Projektarbeit in Teams erfolgreich geübt werden kann.

1 Einleitung

Der Einsatz objektorientierter Technologien hat unbestreitbar Vorteile und ist seit längerer Zeit Stand der Technik in der Industrie. Auch in der Lehre hat die Objektorientierung ihren Platz. Oftmals fehlt es jedoch an deren praktischer und richtigen Umsetzung. Die Ausbildung beschränkt sich auf den Unterricht von OO-Sprachen und grundlegenden OO-Konzepten. Es fehlt jedoch an dem sinnvollen Einsatz der OO-Technologien in praktischen Projekten. Dies mag vielleicht auch daran liegen, dass der Einsatz von OO-Technologien vor allem in den ersten Projektphasen ein sehr intuitiver Prozess ist, der viel Abstraktionsvermögen voraussetzt. Nur nach ausreichender Übung werden OO-Konzepte richtig eingesetzt.
Auf die gleichen Probleme treffen Projektteams in der Industrie, die zum ersten mal auf Objektorientierung umstellen wollen. Auf Grund der Erfahrung von Consulting-Projekten für die Industrie wurde der Einsatz von OO-Konzepten und vor allem des zugehörigen Softwareentwicklungsprozesses in der Lehre vorangetrieben.
Der Studienplan der "Technischen Informatik" am Fachbereich Informatik an der Fachhochschule Ulm sieht im 6. Semester ein Fach mit dem etwas allgemeinen Titel "Allgemeine Informatik" vor. In der Vergangenheit wurden von den Studenten in diesem Fach die klassischen Studienarbeiten als Vorbereitung zur Diplomarbeit erstellt. In Gruppen von ein bis zwei Studenten wurden kleinere Projekte erstellt, deren Qualität jedoch oftmals nicht überzeugen konnte. Laut Studienplan sollen die Studenten in diesem Fach "alles bisher im Studium Gelernte zusammen einsetzen". Dieses Fach bot sich daher an, den Einsatz von OO-Technologien und Softwareentwicklungsprozessen in kleinen Teams zu praktizieren.

2 Vorkenntnisse

Folgende Vorkenntnisse konnten aus dem bisherigen Studium in diesem Zusammenhang vorausgesetzt werden:

- Programmierausbildung (C, C++, 16 SWS)
- Softwaretechnologie (OOA, 2 SWS)
- Datenverarbeitung (8 SWS)
- Betriebssysteme (6 SWS)
- Datenbanken (4 SWS)
- Java (Wahlfach) (4 SWS)

Das Fach "Allgemeine Informatik" wird mit 4 SWS im Studienplan angesetzt.
Was im Studienplan bisher nicht berücksichtigt wurde, ist ein Fach zum Thema "Projektmanagement". Dies wurde jedoch im neuen Studienplan bereits korrigiert.

3 Ziele

Durch die Neuausrichtung des Faches "Allgemeine Informatik" auf konsequenten Einsatz von OO-Konzepten und die Einführung eines definierten Softwareentwicklungsprozesses wurden folgende Ziele verfolgt:

- Verbesserung der Teamarbeit unter den Studenten
- Nutzung von Projektmanagement
- Konsequenter und sinnvoller Einsatz von OO-Konzepten
- Verbesserung der Softwarequalität
- Einführung eines definierten Softwareentwicklungsprozesses

Die Erfahrung nach den ersten Semestern zeigt, dass alle Ziele erreicht wurden. Wie dies im Einzelnen umgesetzt wurde, zeigen die nun folgenden Abschnitte.

4 Organisation

Organisatorisch wurden o.a. Ziele wie folgt umgesetzt:
Zu Beginn des Semesters werden die Studenten in Gruppen zu jeweils 5-6 Personen eingeteilt. Die Einteilung nehmen die Studenten üblicherweise selbst vor. Jede der Gruppen erhält die gleiche Aufgabe in Form eines Ausschreibungstextes, wie er auch in den gängigen Publikationen (z.B. Bundesanzeiger) zu finden ist. Die Studenten wählen sich dann aus den eigenen Reihen einen Projektleiter, der für den organisatorischen Ablauf innerhalb seines Teams verantwortlich ist. Außer der obligatorischen Nutzung eines Softwareentwicklungsprozesses werden zunächst keine weiteren Vorgaben gemacht. Die betreuenden Professoren treten in zwei unterschiedlichen Rollen auf: Kunde des Projektes und Coach der Studenten. Der

Arbeitsaufwand (Deputat) der Professoren, der laut Studienplan mit 2 Stunden veranschlagt wird, beträgt tatsächlich eher 4 Stunden.
Selbstverständlich ist ein regelmäßiges Treffen notwendig. Alle ein bis zwei Wochen berichten die Studenten in einem 20 minütigem Vortrag über den aktuellen Stand des Projektes innerhalb der Gruppe. Die Vorträge werden für jede Gruppe getrennt durchgeführt.
Die Themen der Vorträge sind dabei vorgegeben. Typische Themen sind:

- Vorstellung der Projektorganisation
- Analysemodell
- Persistenz (Datenbankanbindung)
- Designmodell
- Software QM
- Testen

5 Aufgabenstellungen

Wichtig ist sicher die Auswahl eines geeigneten Projektthemas. Das Thema sollte so gewählt werden, dass eine sinnvolle OO-Analyse möglich ist und klein genug, um das Thema innerhalb der knappen Zeit eines Semesters zu bearbeiten. Wichtig ist auch die Abdeckung von typischen Problemstellungen von Projekten wie "Schnittstelle zu externen Systemen" oder "Sicherung der Daten in einer Datenbank (Persistenz)".
Letztes Semester wurde beispielsweise die Aufgabe gestellt, für eine fiktive Firma eine integrierte Datenhaltung für Messgeräte in der Produktion zu erstellen. Ein zentraler Server hält die Daten von Messgeräten und Sensoren, die beim Einschalten eines Messgerätes benötigt werden. Außerdem wird den verschiedenen Abteilungen der fiktiven Firma (Einkauf, Wartung usw.) verschiedene Sichten auf die Daten geboten.

6 Vorgaben

Eine der wichtigsten Vorgaben ist die obligatorische Nutzung eines definierten Softwareentwicklungsprozesses. Wir setzen den "Rational Unified Process" (RUP) ein, der den Studenten in Form der HTML-Seiten des Prozesses zur Verfügung gestellt wird. Außerdem ist das Begleitbuch zum RUP Pflichtlektüre. Der RUP wurde gewählt, da hier schon sehr viele Templates für die zu erstellenden Artefakte als HTML und Word-Template zur Verfügung stehen. Außerdem entspricht der iterative Ansatz des RUP am ehesten der Wirklichkeit in der Praxis.
Der RUP setzt den OO-Ansatz direkt voraus, so dass OO-Techniken auf jeden Fall zum Einsatz kommen. Ebenfalls fest verbunden mit dem RUP ist der konsequente Einsatz der UML. Alle während des Prozesses erstellten Dokumente und auch der Inhalt der Folien bei der Präsentation müssen der UML entsprechen. Einfache

Klötzchengrafiken ohne die korrekte semantische Bedeutung der Grafikelemente sind nicht zugelassen.
Sicher ist es nicht möglich, im Rahmen des Projektes den kompletten RUP einzuführen und zu nutzen, zumal die Studenten sich auch noch mit dem Projektmanagement und der Aufgabenstellung beschäftigen müssen. Sinnvoll wäre es, und dies zeigt auch das Feedback der Studenten, den Entwicklungsprozess mindestens ein Semester vorher einzuführen. Nach anfänglichen Problemen zeigt sich aber, dass die Studenten sich auch ohne Vorkenntnisse erstaunlich schnell mit dem RUP zurecht finden.
Um die Komplexität des RUP dennoch zu bewältigen, haben wir aus dem RUP einige Schlüsselartefakte ausgewählt, die von den Studenten während des Projektverlaufes zu erstellen sind. Diese sind:

- Vision-Document
- Use-Case Dokumente
- Analysemodell
- Software-Architecture Document (Design + Interaction)
- Test-Dokumente

Im Vision-Dokument wird der Inhalt der Aufgabe von den Studenten auf Grund der geführten Interviews mit dem Kunden beschrieben und somit der Umfang der Arbeit festgelegt. Das Vision-Dokument enthält die "stakeholder needs" und stellt somit in Zusammenhang mit den Use-Case-Dokumenten das Pflichtenheft dar, das im RUP nicht direkt vorkommt.
Die Use-Case-Dokumente beschreiben die einzelnen Anwendungsfälle der zu erstellenden Software. Diese Dokumente sind sehr wichtig, damit erstens der Funktionsumfang klarer wird und zweitens diese Anwendungsfälle später in Kollaborations-Diagrammen realisiert werden können.
Ein sehr wichtiges Artefakt ist das Analysemodell. Hier ist erfahrungsgemäß die meiste Hilfestellung nötig, da dafür ein gutes Abstraktionsvermögen notwendig ist. Wie auch in realen Projekten sollte darauf viel Sorgfalt verwandt werden. Evtl. muss das Modell komplett erneuert werden. Um den Studenten Hilfestellung zu geben, werden die folgenden Punkte vorgegeben, auf deren Einhaltung strikt geachtet wird. Erfahrungsgemäß werden dabei die meisten Fehler gemacht:

- Abbildung aller wichtigen Teile der realen Welt in Klassen.
- Klare Aufgaben einer Klasse. Eine Klasse muss eine klar abgegrenzte Aufgabendefinition besitzen.
- Modellierung der Beziehungen zwischen den Klassen als Assoziationen.
- Jede Assoziation muss sinnvoll benannt sein. Nicht nur mit "hat ein"!
- Jede Generalisierung muss klar eine "ist-ein" Relation darstellen.

Gute Hilfestellung bieten Design-Patterns, die bereits im Fach Programmieren III eingeführt werden.

Während des Entwicklungsprozesses wird das Software-Achitecture-Document erstellt. Dieses Dokument enthält das Designmodell sowie die verschiedenen Sichten auf die Software-Architektur:

- Use-Case-View
- Logical-View
- Process-View
- Deployment-View
- Datenmodell

Beim Datenmodell kann ausnahmsweise auf UML verzichtet und ein normales ER-Diagramm präsentiert werden.
Im Testdokument werden schließlich die Testpläne für die zu erstellende Software dokumentiert.
Wichtig erscheint auch, dass sich die Studenten mit dem Thema Software-QM beschäftigen. Auch dies kann im Rahmen eines Semesters sicher nur gestreift werden. Eine sehr einfache Version eines QM ist die Vorgabe von Programmierrichtlinien, die sich die Studenten selbst erstellen. In geringem Umfang ist auch ein Review der erstellten Artefakte möglich. Ein Testplan und die Dokumentation der Tests wird ebenfalls gefordert.

7 Auftretende Probleme

Das Hauptproblem bei der Einführung des RUP unter dem derzeitigen Studienplan ist, das der Prozess während der Benutzung gelernt werden muss. Ein Fach zum Thema Projektmanagement ist vorher notwendig. Im künftigen Studienplan wurde dies berücksichtigt.
Ein anders Problem ist die Stellung des Projektleiters innerhalb des Teams. Da dieser keine a priori Autorität besitzt, d.h. nicht von einer Geschäftsleitung eingesetzt ist, sind Weisungen innerhalb der Gruppe nicht immer möglich. Da die Studenten den Leiter aber aus ihrer Gruppe selber wählen können, wird dieses Problem etwas gemildert.
Der im Studienplan vorgegebene Aufwand (4 SWS, plus impliziertem Faktor zur Vorbereitung) reicht i.A. nicht aus und wird von den Studenten im Eifer des Gefechts oft überschritten.

8 Notengebung

Ein schwieriges Problem am Ende ist die Beurteilung und Benotung der Ergebnisse. Jeder Student soll eine Einzelnote erhalten, obwohl eigentlich ein Projektteam die Arbeit geleistet hat. Hier können und werden natürlich Ungerechtigkeiten auftreten. Entweder gibt es Mitläufer, die eigentlich schlechter sind, oder sehr gute Studenten werden durch ein langsames Team zu schlecht bewertet.

Wir wählen daher folgendes Schema: Die Einzelvorträge werden benotet. Ebenso erhält das Projektteam eine Note für das Gesamtprojekt. Die beiden Noten werden im Verhältnis 1:1 gemittelt. Dies ergibt eine ziemlich gute "Näherung". Im Einzelfall nehmen wir uns dennoch die Freiheit, die Note nach oben oder unten zu korrigieren. Im Laufe der Betreuung der Studenten in der Rolle des Coachs können die Fähigkeiten dann aber doch gut eingeschätzt werden, so dass eine gerechte Benotung möglich ist. Diese Art der Benotung wird von den Studenten allgemein akzeptiert.

9 Ergebnis und Ausblick

Insgesamt lässt sich feststellen, dass die vorgegebenen Ziele erreicht wurden. Ein Ergebnis, das bereits nach einem Semester festgestellt werden konnte, ist, dass die Qualität der Software im Gegensatz zu Zweimann-Projekten deutlich verbessert werden konnte. Dies wurde zwar nicht mit expliziten Metriken gemessen kann jedoch, neben dem natürlich immer vorhandenen subjektiven Eindruck, an folgenden Punkten verdeutlicht werden:

- Die Anforderungen des Projektes wurden mit Hilfe der Use-Cases nachprüfbar realisiert.
- Die Semantik der Grafiken ist durch Nutzung vom UML eindeutig und damit besser lesbar.
- Ein Analysemodell ist vorhanden.
- Die Architektur des realisierten Systems ist durch die vorgegebenen Views klar erkennbar.
- Durch Nutzung der Programmierrichtlinien ergibt sich ein besser lesbarer Code.
- Die Dokumentation ist vollständig.

Dadurch, dass die gleiche Aufgabe auf drei Teams verteilt wurde, entstand auch eine Gruppendynamik durch Konkurrenz der Gruppen untereinander. Jeder Gruppe will besser als die andere sein. Jeder innerhalb der Gruppe identifiziert sich viel stärker mit seinem Projekt als früher bei den Zweimann-Projekten. Auch dies trägt zum Gelingen des Projektes bei.
Nach den positiven Erfahrungen werden wir sicher weitere Projekte dieser Art durchführen. Einige Details können aber dennoch verbessert werden. Beispiel hierfür ist eine klarere Aufgabenstellung oder noch bessere Vorgaben bei den zu erstellenden Artefakten.

10 Empfehlungen

Nach den gemachten Erfahrungen können die folgenden Empfehlungen für derartige Projekte ausgesprochen werden:

- Ein Softwareentwicklungsprozess ist prinzipiell auch für solche Übungsprojekte geeignet. Dabei müssen jedoch die geforderten Schlüsselartefakte vorgegeben werden. Dies sollte möglichst an konkreten Beispielen erfolgen.
- Den Studenten sollten genügend Freiheiten gelassen werden, damit die in der Gruppe vorhandenen Fähigkeiten, z.B. Erfahrung mit Programmiersprachen, genutzt werden können. Dies ist zwar nicht ganz realistisch, aber durchaus tolerierbar.
- Warnung an die Studenten vor unbekannten Risiken. Diese neigen dazu, Technologien einzusetzen, nur weil diese gerade "in" sind, ohne den damit verbundenen Aufwand richtig abzuschätzen.
- Möglichst kleine Aufgaben stellen, auch mit dem Risiko, das Ergebnis nicht real einsetzen zu können. Es kommt beim Erlernen von OO-Techniken viel mehr auf eine gute Analyse bzw. ein gutes Design an, als den ganzen Zeitaufwand in das Erstellen von Codezeilen zu legen.
- Bei der OO-Analyse ist Hilfestellung nötig. Hier finden sich erfahrungsgemäß die meisten Defizite, da eine realistische Analyse bei der Programmierausbildung zu kurz kommt. Das Analysemodell muss mit den Studenten ausführlich besprochen und ggf. korrigiert oder auch neu entworfen werden. Ohne gute Analyse ist der Erfolg des Projektes stark gefährdet.
- Nach Ende des Projektes soll eine Abschlussbesprechung mit den Studenten durchgeführt werden. Damit wird wiederum die Qualität dieses Lehrangebots verbessert.

11 Literatur

Aus folgender Literaturliste zum Thema ist [1] für die Studenten obligatorisch und [2],[11],[12] wird dringend empfohlen:

[1] Philippe Kruchten, The Rational Unified Process, An Introduction, Second Edition, Addison Wesley

[2] Balzert, Lehrbuch der Software Technik, Spektrum

[3] Booch, Rumbaugh, Jacobsen, Das UML Benutzerhandbuch, Addison Wesley

[4] Mark C. Paulk et al, "Key Practices of the Capability Maturity Model - Version 1.1", Software Engineering Institute - Carnegie Mellon University

[5] Terry Quatrain, Visual Modeling with Rational Rose and UML, Addison Wesley

[6] Booch, Objektorientierte Analyse und Design, Addison Wesley

[7] Jacobson, Object-Oriented Software Engineering, A Use-Case driven Approach, Addison Wesley

[8] Der Rational Unified Process, http://www.rational.com

[9] Craig Lamman, Applying UML ans Patterns, An Introduction to Object-Oriented Analysis and Design ,Prentice Hall

[10] Wolfgang Pree, Design Patterns for Object Oriented Software Development, Addison Wesley

[11] Gamma et al., Design Patterns, Elements of Reusable Object-Oriented Software, Addison Wesley

[12] Brown, Whitenack, Crossing Chasm: A Pattern Language for Object-RDBMS Integration, Pattern Languages of Program Design 2