J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

Konzeption und Analyse eines Softwarepraktikums im Grundstudium

Andreas Metzger
Fachbereich Informatik, Technische Universität Kaiserslautern
Postfach 3049, 67653 Kaiserslautern
metzger@informatik.uni-kl.de

Zusammenfassung

In diesem Artikel wird ein Softwarepraktikum im Grundstudium der Informatik an der Universität Kaiserslautern diskutiert, in welchem die systematische Erstellung einer komplexen Ampelsteuerung gefordert war.

Neben der Beschreibung der Aufgabenstellung werden die Ergebnisse des Softwarepraktikums und die gemachten Erfahrungen analysiert, wobei erläutert wird, wie diese Resultate in bestehende Verbesserungen des Praktikums einflossen und welche zukünftigen Veränderungen sinnvoll erscheinen.

1 Einleitung

Im Wintersemester 1998/99 fand an der Universität Kaiserslautern eine Modernisierung des Curriculum der Informatikstudiengänge [1] statt, bei welcher die Inhalte der Grundvorlesungen der Praktischen Informatik stärker in Richtung der ingenieurmäßigen Softwareentwicklung verlagert wurden. In drei jeweils vierstündigen Vorlesungen zu dieser Thematik werden Kenntnisse über Prinzipien, Techniken, Methoden und Werkzeuge für eine systematische Erstellung komplexer Software vermittelt, welche in einem abschließenden Softwarepraktikum mittels einer großen Aufgabe vertieft werden. Für dieses Praktikum werden neben einer Aufgabe im Feld der betrieblichen Informationssysteme und einer Aufgabe zu einem allgemeinen Gebiet, auch eine Aufgabe in der Domäne der eingebetteten Systeme (ES) angeboten. Dieser Artikel beschreibt die Konzeption der ES-Aufgabe [2] und analysiert die gemachten Erfahrungen.

Als Thema für die ES-Aufgabe wurde die Realisierung einer Ampelsteuerung gewählt, da dies eine leicht verständliche und alltägliche Anwendungsdomäne darstellt. Bei der Durchführung dieser Aufgabe werden alle Phasen eines typischen Softwareentwicklungsprozesses durchlaufen – auch wenn einige davon nur angedeutet werden. Am Ende eines kompletten Prozessdurchlaufs schließt sich eine Iteration mit modifizierter Aufgabenstellung an.

In Abschnitt 2 werden zunächst die Inhalte der ES-Aufgabe vorgestellt und von den, bezüglich der Inhalte, gemachten Erfahrungen berichtet. Anschließend wird in Abschnitt 3 die Durchführung des Praktikums beschrieben und einige Ergebnisse und Konsequenzen dargestellt.

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

2 Die Praktikumsaufgabe

Aufgabe ist die Entwicklung einer Ampelsteuerung für einen sechs Ampelkreuzungen umfassenden Ausschnitt aus dem Straßennetz von Kaiserslautern. Das zu entwickelnde System soll sowohl die lokale Steuerung der Ampeln *einer* Kreuzung, als auch die koordinierte Steuerung *aller* Ampeln in einer „Grünen Welle“ ermöglichen. Desweiteren soll das System über eine Benutzerkonsole verfügen.

Den Studierenden wurden zu Beginn die Anforderungen an das System aus Kundensicht und die Straßenpläne der Innenstadt ausgehändigt. Desweiteren erhielten sie Auszüge aus einem domänenspezifischen Handbuch. Die Entscheidung, die Aufgabe für einen konkreten Straßenzug aus der Stadt der Hochschule durchführen zu lassen, hat sich äußerst positiv auf die Motivation der Praktikumsteilnehmenden ausgewirkt.

Zusätzlich zu obigen Dokumenten wird für jede Phase der Systementwicklung zusätzlich eine Einführung in die zu verwendende Modellierungs-, bzw. Implementierungstechnik gegeben und jeweils erprobte Methoden für die Anwendung dieser Techniken vorgestellt. Die dabei präsentierten Inhalte werden in den folgenden Unterabschnitten skizziert.

2.1 Analyse und Entwurf

Während der Analysephase findet zunächst eine Modellierung der Struktur und des Verhaltens des Systems mit Hilfe von UML [3] statt. Neben einem Ausgangspunkt für die anschließende Implementierung dienen diese Modelle zusätzlich der Dokumentation. Deshalb wird von den Studierenden verlangt, diese Modelle stets auf dem aktuellen Stand zu halten.

Die Struktur des Systems soll durch ein Klassen- und ein Instanzenmodell beschrieben werden. Als Methode zur Aufstellung solcher Modelle wird vorgeschlagen, die physikalischen Objekte (Sensoren, Aktuatoren, Kontroller, etc.) als Ausgangspunkt zu nehmen, da sich dieser Ansatz in einer in unserer Arbeitsgruppe entwickelten Anforderungsanalysemethode bereits bewährte [4]. Zusätzlich erlaubt diese Modellierung von Klassen in der Analysephase eine frühe Festlegung von Entwicklerverantwortlichkeiten.

Um Testfälle für den späteren Test des Systems zu erhalten, sollen während der Analyse Szenarien aufgestellt werden, die ausgewählte Aspekte der gewünschten Systemfunktionalität beschreiben. Diese Szenarien dienen zudem als Einstieg für die anschließende Verhaltensmodellierung. Die Modellierung solcher Szenarien erfolgt mit Hilfe von UML-Sequenzdiagrammen, wobei zur Kommunikation asynchrone Nachrichten, welche typisch für eingebettete Systeme sind, Verwendung finden. Nach der Aufstellung dieser Sequenzdiagramme soll das Verhalten dann vollständig mit Hilfe endlicher Automaten in UML-Zustandsdiagrammen spezifiziert werden.

Ein Entwurf (Design) im eigentlichen Sinne kann in diesem Praktikum aus Zeitgründen nicht durchgeführt werden, da dazu z. B. die endgültige Zielhardware und nicht-funktionale Anforderungen wie z. B. Verlässlichkeit berücksichtigt werden müssten. Daher entwickeln die Studierenden einen Prototyp der Ampelsteuerung.

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

Als Modellierungswerkzeug wird die kommerzielle Telelogic Tau SDL Suite [5] zur Verfügung gestellt. Dieses Tool eignet sich insofern zur UML-Modellierung, dass auch Editoren für die wichtigsten UML-Diagrammtypen vorhanden sind.

Ergebnisse

Die Modellierung des Systems wurde von den Studierenden ohne schwerwiegende Probleme durchgeführt, da die UML-Notation aus den vorausgegangenen Vorlesungen gut verstanden war. Die abgegebenen Klassendiagramme hatten im Schnitt eine Zahl von 11 Klassen und 15 Relationen mit nur geringen Abweichungen zwischen den Gruppen. Diese relativ einheitliche Komplexität (im Vergleich zu Praktika wie z. B. [6]) liegt sicherlich an der Vorgabe, sich an den physikalischen Objekten zu orientieren. Die Zustandsautomaten besaßen eine mittlere maximale Komplexität von 54 Zuständen und 88 Transitionen.

Die Entscheidung, ein kommerzielles Werkzeug an Stelle freier Software einzusetzen, hat sich in der hohen Qualität der abgegebenen Modelle bezahlt gemacht. Viele der freien Tools, die wir in den vorausgegangenen Jahren eingesetzt hatten, verfügten oft über keinerlei Syntax-Prüfungen oder ermöglichten keine Ausgabe der Modelle in einem vernünftigen Format. Der Umgang mit dem Werkzeug konnte von den Studierenden schnell erlernt werden, vermutlich auch weil nur die Editor-Funktionalität Verwendung fand und keine komplexeren Funktionen benutzt wurden.

2.2 Implementierung und Test

Die Implementierung erfolgt in der Programmiersprache Java [8], wobei sehr großen Wert auf eine *sinnvolle* Dokumentation des Codes gelegt wird [9].

Am Anfang der Implementierung wird eine Abbildung der Modellklassen auf Klassen der Programmiersprache vorgenommen und die Relationen zwischen diesen Klassen umgesetzt. Daran schließt sich die Implementierung der Zustandsdiagramme an. Dabei wird von einer prozeduralen Realisierung mittels bedingter Anweisung (Case- oder If-Then-Else-Konstrukte) abgeraten, weil diese Lösung zu einem sehr unübersichtlichem und schlecht erweiterbarem Code führt. Als objektorientierter Realisierungsansatz wird deshalb das State-Pattern [10] propagiert.

Bei der Implementierung der grafischen Benutzersschnittstelle hatten die Studierenden außer der Vorgabe, das Model-View-Controller-Pattern [10] anzuwenden, weitgehend freie Hand.

Die Verbindung der Ampelsteuerung mit der Umgebung wird mittels der Java-RMI-Technik [11] aufgebaut. Da die Ampelsteuerung verständlicherweise nicht mit der echten Umgebung getestet werden konnte, wurde den Studierenden eine grafische Simulation der Umgebung zur Verfügung gestellt, welche neben dem Verhalten der Ampeln auch eine dynamische Anzahl an Fahrzeugen simuliert.

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

Ergebnisse

Bei der Beurteilung der Abgaben hat sich gezeigt, dass viele der OO-Konzepte noch nicht vollständig von den Studierenden beherrscht wurden. So hatten z.B. einige Gruppen erhebliche Mengen von Code produziert, was durch den Einsatz von Vererbung nicht nötig gewesen wäre. Dies dürfte wohl stark mit der Art der Übungen der vorausgegangenen Vorlesungen zusammenhängen, bei welchen solche Überlegungen auf Grund der geringen Größe der Aufgaben nicht nötig waren oder sich nicht so deutlich in einem erhöhten Aufwand niederschlugen. Diese Erfahrung zeigt deutlich, wie wichtig die Vertiefung der Kenntnisse an einem großen Beispiel ist. Auch in [12] kommt man zu diesem Fazit.

Desweiteren gab es bei der Umsetzung der Pattern Probleme. Diese wurden zu stark im Sinne von Templates verstanden, weshalb die Übertragung der eigentlichen Lösungsidee auf die konkrete Aufgabenstellung zum Teil fehlschlug. Mögliche Ursachen für diese Problematik können Schwierigkeiten bei der Vermittlung objektorientierter Programmierung (siehe [12]) oder ein mangelnder Unterricht in Mustern sein (siehe [14]). Man kann dies sicherlich verbessern, indem in den dem Praktikum vorausgehenden Vorlesungen der Charakter der Pattern besser herausgearbeitet wird.

Die Größe des von den Gruppen jeweils abgegebenen Codes lag zwischen 4400 und 15000 Zeilen (im Schnitt bei 8000 Zeilen) inklusive der Kommentare. Obwohl die Anforderungen an die GUI sehr moderat formuliert waren, investierten dennoch viele Gruppen großen Aufwand in diesen Teil. Die Komplexität der GUI erreichte in einigen Fällen die Komplexität der eigentlichen Steuerung. Im Durschnitt betrug der Anteil des GUI-Codes am Gesamt-Code 30%. Hier sollte über eine genauere Vorgabe – besonders auch dessen was *nicht* gefordert ist – nachgedacht werden, um die Verteilung des Arbeitsaufwandes auf die gewünschten Schwerpunkte zu lenken.

2.3 Iteration

Im Softwarepraktikum sollen die Studierenden nicht nur eine Entwicklung von Null durchführen, sondern auch die Probleme bei einer Erweiterung oder Änderung eines bestehenden Systems erkennen. Daher wurde am Ende des Praktikums eine Iterationsphase angeschlossen, für welche die Umsetzung einer geänderten Problembeschreibung gefordert war. Typische Änderungen waren das Hinzufügen einer zusätzlichen Ampel und die Veränderung der Hauptverkehrsrichtung der „Grünen Welle“.

Ergebnisse

Da die Iteration schon zu Beginn des Praktikums angekündigt wurde, stellte sich diese Phase als unproblematisch für die meisten Gruppen dar. Lediglich der Aufwand für die Anpassung des Systems variierte. So gab es Gruppen, die ihre Steuerung sehr generisch ausgelegt hatten während andere nicht so starken Wert auf Generizität gelegt hatten. Die durchschnittliche Zunahme des Codes lag bei 2500 Zeilen. Das führte zu einer durchschnittlichen Gesamtkomplexität des am Ende des Praktikums verfügba-

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

ren Codes von 10000 Zeilen. Im Vergleich zu anderen Praktika (z.B. [6] oder [7]) scheint diese Komplexität relativ hoch, weshalb man, auch unter Berücksichtigung der in Abschnitt 3 vorgestellten Resultate, über eine Vereinfachung der Aufgabe nachdenken sollte.

3 Die Durchführung des Praktikums

Für die Durchführung des Praktikums wurden zwölf Wochen veranschlagt, in welchen Gruppen von jeweils sechs Personen die einzelnen Phasen der Softwareentwicklung wie in Abbildung 1 gezeigt durchlaufen. Dabei beinhaltet die Phase der Iteration auch die Vorbereitung auf eine Abschlusspräsentation des Systems. Hier wird neben einer 20-minütigen Systemvorführung durch die ganze Gruppe, von jedem Gruppenmitglied auch ein kurzer, ca. fünf-minütiger Vortrag zu einem ausgewählten Gebiet der Praktikumsaufgabe verlangt.

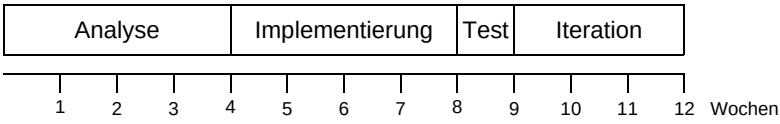

Abbildung 1: Verteilung der Phasen auf die Laufzeit des Praktikums

Das Softwarepraktikum ist als halbtägiges Praktikum geplant, weshalb für den Bruttoarbeitsaufwand der Studierenden von 12 Semesterwochenstunden ausgegangen wird. In diesem Aufwand sind neben der Zeit für das eigenständige Arbeiten in den Gruppen, die Vorstellungen der jeweiligen Aufgaben durch die Betreuer (ca. 30 Min. pro Woche) und sogenannte Präsenzzeiten (mind. eine Stunde pro Woche) enthalten. Die Studierenden werden dabei in den Präsenzzeiten permanent von Mitarbeitern oder Hilfskräften betreut, wodurch Fragen und Probleme sofort diskutiert werden können.

Als Motivationsschub für das Praktikum wird am Ende ein kleiner Wettbewerb durchgeführt, in welchem die Steuerungssysteme der Gruppen nach drei Kriterien beurteilt werden und die jeweilige Gewinnergruppe einen kleinen Preis und Urkunden erhält.

Ergebnisse

Die Einteilung in die in Abbildung 1 gezeigten Phasen hat sich als günstig erwiesen. Lediglich die gegen Ende des Semester angesetzte Systemvorführung hat sich als problematisch herausgestellt, da wenige Tage später die Semesterabschlussklausuren anderer Veranstaltungen stattfanden. Da für die Einzelvorträge nur jeweils fünf Minuten blieben, ist für die Zukunft geplant, die Vorträge über das ganze Semester verteilt und dann jeweils über ca. 20–30 Minuten halten zu lassen. Dies kann z.B. sinnvoll im Rah-

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

men der Präsenzzeiten erfolgen. Auch um die Präsentationsfähigkeiten der Studierenden zu verbessern scheint eine solche Erweiterung äußerst sinnvoll (siehe [15]).

Als vorteilhaft hast sich die Gruppengröße von sechs Studierenden herausgestellt, da dadurch die sinnvolle Aufteilung der Arbeit innerhalb der Gruppe auf Teams von zwei Personen möglich wird. Desweiteren erfordert diese Zahl an Gruppenmitgliedern eine organisierte Kommunikation. Ein Team von einem Mitarbeiter und einer wissenschaftlichen Hilfskraft konnte jeweils drei solcher Gruppen gut betreuen.

Zur Einführung von Präsenzzeiten entschlossen wir uns, da ein großer Anteil der Studierenden nur noch am heimischen PC arbeitete und der Kontakt zu den Betreuern ausschließlich bei kritischen Problemen gesucht wurde. Da dadurch der typische Praktikumscharakter verloren ging, versuchten wir mit den Präsenzzeiten einen Kompromiss zu finden. Desweiteren sahen wir darin eine sinnvollere Möglichkeit, die Individualleistung einzelner Teilnehmer zu beurteilen, als dies durch das bisherige Testieren möglich war. Das Resultat dieser Änderung wird in Abschnitt 3.1 diskutiert.

Die Durchführung des Wettbewerbs am Ende wurde von den Studierenden positiv aufgenommen. Interessant wäre, wenn man durch diesen Wettbewerb einen Zusatznutzen für den Lernerfolg der Teilnehmer erzielen könnte. Anregungen dazu finden sich in [16].

Erstmalig in diesem Jahr führte die Fachschaft Informatik eine Praktikumsumfrage durch, in welchem 40% der Studierenden den Gesamteindruck des Praktikums mit „sehr gut“ bis „gut“ bewerteten und 35% dem Praktikum ein „befriedigend“ gaben. Als Kritikpunkte wurden von dem jeweils in Klammern angegebenen Anteil an Studierenden genannt:

1. sehr hoher Arbeitsaufwand von mehr als zwölf Stunden (45%)
2. ungenaue Formulierung der Aufgabenstellung und der Dokumentation (30%)
3. hoher Schwierigkeitsgrad der Aufgaben (25%)

Bei der Beurteilung des ersten Punktes sollte man berücksichtigen, dass für viele der Teilnehmer die Alternative zum Mehraufwand das Nichtbestehen gewesen wäre. Dennoch sollte eine Reduzierung der Aufgabe (durch z.B. Verringerung der Anzahl betrachteter Kreuzungen) in Betracht gezogen werden, da eine Reduktion des Anteils der GUI am Gesamtsystem (siehe Abschnitt 2.2) zwar eine Verbesserung bringen könnte, aber vermutlich nicht in dem Maße wie nötig. Bei den Punkten 2 und 3 kommt sicherlich die in Abschnitt 2.2 erwähnte Problematik des Umgangs mit OO-Konzepten und Pattern zum tragen. Dieses wird momentan durch eine Umstellung der Vorlesungen angegangen.

Positiv wurde von den Studierenden der Praxisbezug bewertet. Desweiteren wurde die Arbeit im Team und die Anwendung des in den vorausgegangenen Vorlesungen gelernten Stoffes als Pluspunkte des Praktikums angegeben.

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

3.1 Benotung

Die Gesamtleistung eines jeden Teilnehmers setzt sich aus 50% Gruppenleistung und 50% Individualleistung zusammen. Zur Gruppenleistung tragen die Abgaben zu den einzelnen Aufgaben und die Abschlusspräsentation bei. In die Individualbewertung fließt die Leistung während der Präsenzzeiten und die Qualität des Einzelvortrags ein.

Ergebnisse

Abbildung 2 zeigt einen Vergleich der Noten der Jahre 2001 und 2002. In unseren Augen ergab sich die deutliche Notenverbesserung (um im Schnitt 0,7 Notenstufen) durch Einführung der Präsenzzeiten.

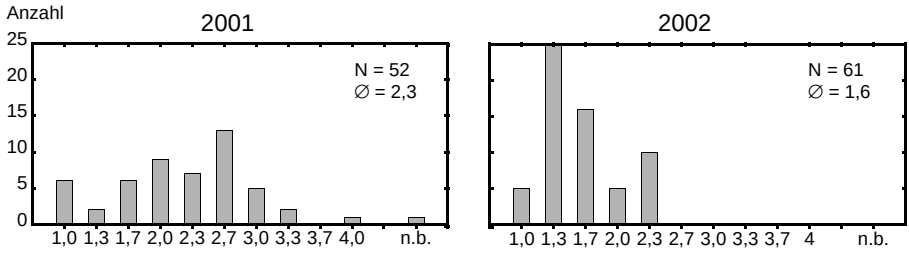

Abbildung 2: Notenverteilung

Neben der deutlichen Verbesserung der Noten aller Teilnehmer reduzierte sich zudem die Variation der Noten innerhalb der einzelnen Gruppen, wie Abbildung 3 zeigt. Diese Tatsache wirkt sich mit Sicherheit positiv auf den bleibenden Eindruck des Praktikums für die einzelnen Teilnehmer aus.

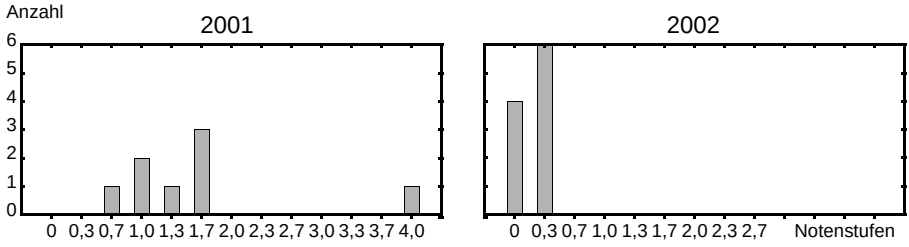

Abbildung 3: Maximale Abweichung der Noten innerhalb einer Gruppe

Wie in [16] richtig formuliert wird, sollte man an dieser Stelle aber die Frage stellen, ob eine Individualbewertung in einem Teampraktikum überhaupt sinnvoll ist. Mit den obigen Resultaten könnte man nun auch umgekehrt argumentieren und aus der sehr geringen Abweichung der Noten innerhalb einer Gruppe folgern, dass eine Individualnote nicht benötigt wird. Die Frage, ob die Notenverbesserung auch wirklich eine Verbesserung des Wissens der Teilnehmenden ergeben hat, lässt sich mit den vorliegenden Ergebnisse leider nicht beurteilen.

J. Siedersleben, D. Weber-Wulff (Hrsg.) (2003): Software Engineering im Unterricht der Hochschulen, SEUH 8, Berlin, dpunkt.verlag, Heidelberg

4 Schlussbetrachtungen

Das Softwarepraktikum im Grundstudium hat sich als eine sinnvolle Bereicherung des Curriculums der Informatikstudiengänge an der Universität Kaiserslautern herausgestellt. Wie an Hand der qualitativen und quantitativen Ergebnisse aufgezeigt wurde, kann und sollte eine stetige Verbesserung der Güte eines solchen Praktikums durch die Berücksichtigung der Ergebnisse vorausgegangener Jahre erzielt werden.

Zum Schluss gilt mein Dank allen Studierenden, Mitarbeitern und Dozenten, welche eine solch erfolgreiche Durchführung des Softwarepraktikums ermöglichten.

Literatur

1. Bunke, B.; Faber, K.; Kirchner, R. et al.: Informatik-Studium. Studienführer Informatik, Universität Kaiserslautern, 1999
   http://www.informatik.uni-kl.de/studium/studienfuehrer/html_noframe/node16.html
2. Metzger, A.: Softwarepraktikum – Teil: Eingebettete Systeme. Web-Seite, Universität Kaiserslautern, 2002
   http://wwwagz.informatik.uni-kl.de/courses/ss02/SWP
3. Booch, G; Rumbaugh, J; Jacobson, I: The Unified Modeling Language User Guide. Addison-Wesley Longman, Reading, Mass., 1999
4. Metzger, A.; Queins, S.: Specifying Building Automation Systems with PROBAnD, a Method Based on Prototyping, Reuse, and Object-orientation. Hofmann, P.; Schürr, A. (Hrsg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems. GI-Edition, Lecture Notes in Informatics (LNI), P-5, Köllen Verlag, Bonn, 2002, S. 135–140
5. Telelogic Tau SDL Suite. Web-Seite, Telelogic, AB, Schweden, 2002
   http://www.telelogic.com/products/tau/sdl/index.cfm
6. Demuth, B.; Hußmann, H.; Zschaler, S. et al.: Erfahrungen mit einem frameworkbasierten Softwarepraktikum. Dreher, B.; Schulz, Ch.; Weber-Wulff, D. (Hrsg.): Software Engineering im Unterricht der Hochschulen SEUH ‘99, B.G. Teubner, Stuttgart, 1999, S. 21–30
7. Bruegge, B.: From Toy Systems to Real System Development: Improvements in Software Engineering Education. Hußmann, H.; Paech, B. (Hrsg.): Software Engineering im Unterricht der Hochschulen SEUH ‘94, B.G. Teubner, Stuttgart, 1994, S. 62–72
8. Eckel, B: Thinking in Java. 2nd Edition. Prentice Hall, Upper Saddle River, New Jersey, 2000.
9. Vermeulen, A; Ambler, S.W.; Bumgardner, G. et al.: The Elements of Java Style. Cambridge University Press, Cambridge, 2000
10. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Mass., 1995.
11. Java Remote Method Invocation Specification. Online-Dokument, Sun Microsystems, 2002
    ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf
12. Schneider, K.: Auf der Suche nach maßgeschneiderten Unterrichtsformen: Das angeleitete Praktikum. Raasch, J.; Bassler, T. (Hrsg.) : Software Engineering im Unterricht der Hochschulen SEUH ‘93, B.G. Teubner, Stuttgart, 1993, S. 66–77
13. Reus, B: Software-Entwicklung im Grundstudiumspraktikum: nein – ja – Java? SEUH ‘99, S. 45–54
14. Engelien, M.; Vocke, H.; Wunsch, R.: Zum Spannungsausgleich zwischen objektorientierter und herkömmlicher Softwareentwicklung. Spillner, A.; Breymann, U. (Hrsg.): Software Engineering im Unterricht der Hochschulen SEUH ‘95, B.G. Teubner, Stuttgart, 1995, S. 109–118
15. Weber-Wulff, D.: Teambildung in Programmierung und Software-Engineering Kursen. SEUH ‘95, S. 82–89
16. Hornecker, E.: Teamtraining und Präsentationstechniken für das Software-Engineering-Praktikum, SEUH ‘95, S. 69–81