Funktionell und verständlich programmieren – so lernen es die Passauer

Andreas Zeller
Universität Passau
Lehrstuhl für Software-Systeme
zeller@acm.org

Zusammenfassung. Das *Praktomat-Programm* ermöglicht Programmierpraktika mit automatischem Testen und rechnergestützten, gegenseitigen Reviews.
Schlüsselwörter: Software-Testen, Software-Review, Programmierausbildung

1 Qualitätskontrolle im Programmierpraktikum

Ich habe einige Jahre Programmierpraktika betreut. Für die Programme, die Studenten bei mir einreichten, galt:

1. Läßt sich das Programm übersetzen, hat es Fehler.
2. Läuft es korrekt, kann man es kaum lesen.
3. Läuft es und ist es verständlich, ist es abgeschrieben.

— Mitarbeiterin, Uni Passau (1998)

Können unsere Studenten tatsächlich programmieren? Sie können: Schließlich haben sie alle ein Programmierpraktikum absolviert. Aber: Können wir sicher sein, daß ihre Programme einen größeren Test bestehen würden? Haben wir uns überzeugt, daß diese Programme verständlich und wohldokumentiert geschrieben sind? Und: Wissen wir, daß ein Programm auch tatsächlich von demjenigen stammt, der uns vorgibt, der Autor zu sein?

Wir wollen, daß unsere Studenten gut programmieren lernen, und wir glauben, daß sich ihr Können in angemessener Programmqualität ausdrückt. Jede dieser Fragen kann damit durch angemessene *Qualitätskontrolle* beantwortet werden. Qualitätskontrolle – das heißt Personaleinsatz: Hiwis und Mitarbeiter, die (neben der selbstverständlichen Beratung) Programme testen und gegenlesen. Dieser Aufwand kostet Geld, weshalb man zwischen Programmqualität und Personaleinsatz abwägen muß: je höher die Anforderungen, desto größer der Aufwand in der Qualitätskontrolle.

Damit sind Wege gefragt, den Aufwand zu verkleinern und gleichzeitig den Lerneffekt, ausgedrückt durch die Programmqualität, zu maximieren. Einen solchen Weg sind wir an der Uni Passau gegangen, und zwar durch die *weitgehende Automatisierung der Qualitätskontrolle im Programmierpraktikum.*

Wie sieht Qualitätskontrolle in der Praxis aus? Die wichtigsten Verfahren der Software-Prüfung sind *Testen* und *Software-Inspektion*, also das Gegenlesen eingereichter Programme. Diese Techniken haben wir automatisiert, in unser Programmierpraktikum integriert und damit nicht nur die Programmqualität gesteigert, sondern zusätzlich noch den Studenten neben dem *Schreiben* von Programmen auch das *Lesen* fremder Programme beigebracht – und das alles bei gleichbleibendem Betreuungsaufwand.

2 Praktomat – eine automatisierte Praktikumsverwaltung

Software ist ein herrliches Feld
um sich zu betätigen
falls es Sie nicht stört daß Prüfen
nicht immer so
viel Spaß macht
falls Sie ein Absturz nicht stört
dann und wann
gerade wenn alles so schön läuft

— Karol Frühauf, Jochen Ludewig und Helmut Sandmayr, *Software-Prüfung*

In der Passauer Informatik müssen die Studenten im zweiten Semester ein Programmierpraktikum absolvieren, in dem sie vier bis fünf Programmieraufgaben mit wachsendem Schwierigkeitsgrad lösen – z.B. Sortierverfahren, Graphensuche, balancierte Suchbäume. Grundlage des Passauer Programmierpraktikums ist

Praktomat, eine eigens entwickelte Praktikumsverwaltung. Im wesentlichen ist Praktomat eine Datenbank, die die Stammdaten und Lösungen der Studenten entgegennimmt und verwaltet. Praktomat kann jedoch noch mehr:

Lösungen werden automatisch getestet.
: Erst wenn ein Programm eine Reihe von Testfällen erfüllt, wird es als Lösung angenommen. Das Testergebnis dient auch der späteren Bewertung.

Studenten können ihre Programme gegenseitig kommentieren.
: Das Kommentieren dient sowohl demjenigen, der den Kommentar erstellt (durch das Lesen und Verstehen eines fremden Programms) als auch demjenigen, der den Kommentar erhält (als Rückmeldung über die Qualität des eigenen Programms).

Jeder erhält eine eigene Aufgabenstellung.
: Dies erzwingt individuelles Arbeiten, indem das platte Abschreiben fremder Programme *de facto* unmöglich gemacht wird. Erst so wird gegenseitiges Kommentieren möglich.

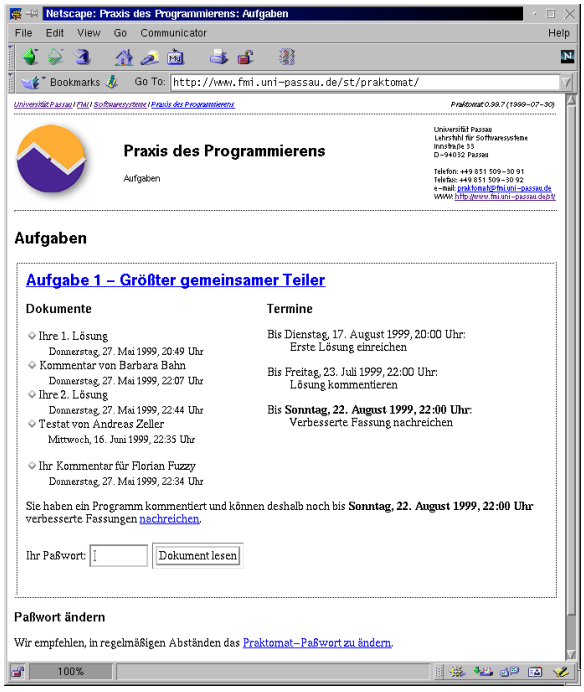

Abbildung 1: **Praktomat-Einstiegsseite. Die Studenten können auf ihre individuellen Aufgabenstellungen per WWW zugreifen. Auch Lösungen werden über das Netz eingereicht.**

Praktomat ist über das WWW zugänglich. Damit können Studenten von überall her und rund um die Uhr auf Praktomat-Dienste zugreifen (Abbildung 1). Auch die Betreuer nutzen Praktomat, um Aufgabenstellungen abzuspeichern und zu testen, Studentendaten zu verwalten und um die Programme schließlich zu testieren.

3 Automatisches Testen

Öffentliche Tests
```
Test summary
# of expected passes 25
# of failures 2
```
Ihr Programm kann so nicht angenommen werden. Bitte untersuchen Sie, warum Ihr Programm die Testfälle nicht erfüllt und versuchen Sie es noch einmal.

— Praktomat, *Programm einreichen*

Damit Praktomat ein Programm annimmt, muß es sich übersetzen lassen und einige öffentliche Testfälle bestehen. Praktomat nimmt den Quellcode entgegen, übersetzt ihn und testet das entstehende Programm. *Schlägt die Übersetzung oder ein Testfall fehl, verweigert Praktomat die Annahme*; die Details werden dem Einreicher mitgeteilt, so daß er das Problem beheben kann (Abbildung 2 auf der nächsten Seite).

Als Testrahmen benutzt Praktomat das DEJAGNU-Programm, das eine textuelle Benutzerinteraktion simuliert. Aus Sicherheitsgründen führt Praktomat das untersuchte Programm in einem „Sandkasten“ aus – einem geschützten Bereich, aus dem heraus keine Datei- oder Netzzugriffe möglich sind. Tatsächlich kann das Programm nur über seine Standard-Eingabe und -Ausgabe interagieren; graphische Oberflächen müssen als (von Praktomat ungetestete) separate Teile realisiert werden.

Da die Programmausgabe maschinell verarbeitet wird, muß die Aufgabenstellung das Verhalten des Programms in Form seiner Ein- und Ausgabe exakt beschreiben – was zu ungewöhnlich detaillierten und präzisen Aufgabenstellungen führt. Genauso präzise müssen aber auch die eingereichten Lösungen das geforderte Verhalten realisieren. Eine gegebene Aufgabenstellung wortgetreu ohne schmückendes Beiwerk zu realisieren – dies ist eine wichtige Erfahrung und ein großer Lerneffekt für die Studenten.

Neben den öffentlichen Tests werden beim Einreichen noch *geheime Testfälle* durchgeführt, die den Studenten nicht mitgeteilt werden. Ihr Ergebnis hat keinen

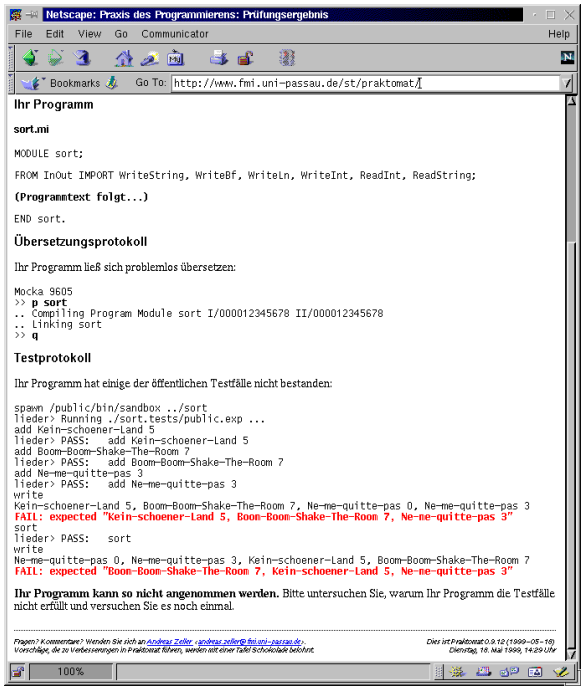

Abbildung 2: Automatisches Testen. Praktomat übersetzt das eingereichte Programm und führt eine Reihe von Testfällen aus. Passen die Ausgaben nicht ins erwartete Muster, schlägt der Test fehl; das Programm muß korrigiert und neu eingereicht werden.

Einfluß auf das Einreichen, wird später aber beim Testieren der Programme durch Mitarbeiter berücksichtigt. Die Studenten wissen natürlich, daß es geheime Testfälle gibt, und werden so ermuntert, ihr Programm über die öffentlichen Testfälle hinaus mit selbst entworfenen Testfällen systematisch zu prüfen.

4 Individuelle Aufgabenstellungen

— Ich habe keine Lust mehr. Kann ich nicht mal Dein Programm sehen?

— Ja schon, aber Du hast Mergesort. Ich hab' Quicksort.

— Studenten des Programmierpraktikums

Mit Praktomat wollen wir individuelles Arbeiten erzwingen, sprich: Abschreiben vermeiden. Zu diesem Zweck weist Praktomat jedem Studenten eine *individuelle Aufgabenstellung* zu. Dies klingt aufwendiger, als es ist: tatsächlich gibt es nur eine Aufgabenstellung, die aber an entscheidenden Stellen *parametrisiert* ist. So wird etwa die Aufgabenstellung

```
Als Datenstruktur benutzen Sie
ifdef(V1, einen AVL-Baum, einen Rot-
Schwarz-Baum)
```

von der einen Hälfte der Studenten einen AVL-Baum als Datenstruktur verlangen, von der anderen einen Rot-Schwarz-Baum. Insgesamt gab es bis zu 8 solcher Konfigurationsvariablen und damit 256 verschiedene Aufgabenstellungen.

Neben Unterschieden in den elementaren Datenstrukturen haben wir auch Unterschiede im *externen Verhalten* des Programms gefordert. Damit konnte ein fremdes Programm nicht die eigenen (ebenfalls parametrisierten) Tests bestehen – kurz, wer ein fremdes Programm übernehmen wollte, mußte eine nicht unbeträchtliche Anpassungsarbeit leisten. Diese Arbeit ist jedoch vergleichbar mit der Anpassung eines Algorithmus' aus einem Lehrbuch – und lehrreich genug, die Lernziele des Praktikums zu erfüllen.

Was wir nicht ausschließen konnten, ist, daß ein Student A sich sein Programm vollständig von einem anderen Studenten B erstellen läßt. Das halten wir aber für unwahrscheinlich: Da B eine eigene Aufgabenstellung hat, kann er nicht A sein Programm überlassen (was nicht verwerflich ist), sondern müßte selbst für A dessen Aufgabenstellung realisieren, sich also wissentlich an A's Täuschungsversuch beteiligen. Dies setzt aber ein hohes Maß an krimineller Energie voraus – genauso könnte B auch A's Studien- oder Diplomarbeit schreiben. Stichproben können hier abschreckend wirken.

5 Programme kommentieren

Sie können den obigen Programmtext am linken Rand mit Kommentaren versehen, um auf bestimmte Punkte aufmerksam zu machen:

f – Funktionalität
d – Dokumentation
e – Einrückung
s – Funktionsstruktur
b – Bezeichner
l – Lokalität
a – Anpaßbarkeit

Beispiel: „d?“ steht für problematische oder fehlende Dokumentation, „a!“ für herausragende Anpaßbarkeit.

— Praktomat, *Programm kommentieren*

Aus Sicht der Studenten war der aufregendste und lehrreichste Teil von Praktomat das *gegenseitige Kommentieren*. Nach Ablauf einer ersten Frist, in der die eigene Lösung eingereicht sein mußte, konnte ein Student

die Lösung eines Kommilitonen („Partners“) zum Kommentieren abrufen. Dabei stellte Praktomat sicher, daß

- der Student eine eigene Lösung eingereicht hatte, und
- die Aufgabenstellungen der Partner sich unterschieden.

Das Kommentieren war nicht Pflicht; wohl aber konnte ein Student durch das Abgeben eines Kommentars seine Abgabefrist um eine Woche verlängern (um so das beim Kommentieren erworbene Wissen ins eigene Programm einfließen zu lassen.)

Beim Kommentieren wurde dem Studenten der Quelltext seines Partners präsentiert, in dem er nach Belieben Kommentare anbringen konnte. Um die Kommentare zu strukturieren, haben wir eine Reihe von *Bewertungsfragen* gestellt, die die Studenten beantworten sollten – wobei sie für ihre Bewertungen im Code Belege markieren konnten (Abbildung 3).

Funktionalität

- Erfüllt das Programm die Aufgabenstellung?

  A – ja; ich würde dies weiterempfehlen

  B – ja; aber ich würde mir kleine Verbesserungen wünschen

  C – nein; nur wenn zahlreiche Verbesserungen vorgenommen werden

  D – nein; ich würde davor warnen

  (In dieser Bewertung bin ich mir sicher / eher sicher / eher unsicher / unsicher)

Beispiele, bei denen sich das Programm nicht gemäß der Aufgabenstellung verhält: . . .

— Praktomat, *Funktionalität kommentieren*

Die erste Frage betraf die *Funktionalität*. Wohl wußten die Studenten, daß das Programm die öffentlichen Testfälle bestanden hatte. Zu unserem Erstaunen beließen es viele Kommentierer nicht beim Durchlesen des Codes, sondern kopierten den Quellcode aus dem Textfenster, um ihrerseits das Programm zum Laufen zu bringen und zu testen. Ein Kommentieraufwand von mehreren Stunden war keine Seltenheit und zeugt von großem Enthusiasmus.

Für die Bewertung konnten die Studenten auf Werte von A bis D zurückgreifen, wie oben beschrieben. Die Werte und ihre Bedeutung wurden an klassische Bewertungsschemata im wissenschaftlichen *peer review* angelehnt („will fight for acceptance“, „will agree with acceptance“, „will agree with rejection“, „will fight for rejection“); ein neutraler Mittelwert wurde bewußt vermieden, um die Meinungsbildung zu fördern.

Verständlichkeit

- Sind die Komponenten und Verfahren gut dokumentiert? (A / B / C / D)
- Sind die Bezeichner aussagekräftig und konsistent gewählt? (A / B / C / D)
- Kann das Programm ohne großen Aufwand an veränderte Randbedingungen (z.B. durch Ändern von Konstanten) angepaßt werden? (A / B / C / D)

  . . .
- Gesamteindruck: Kann die Lösung als Muster in einem Lehrbuch verwendet werden? (A / B / C / D)

— Praktomat, *Verständlichkeit kommentieren*

Neben der Funktionalität stand die *Verständlichkeit* des Programms im Mittelpunkt; anhand eines einfachen *style guides* konnten die Studenten sechs Kriterien wie die

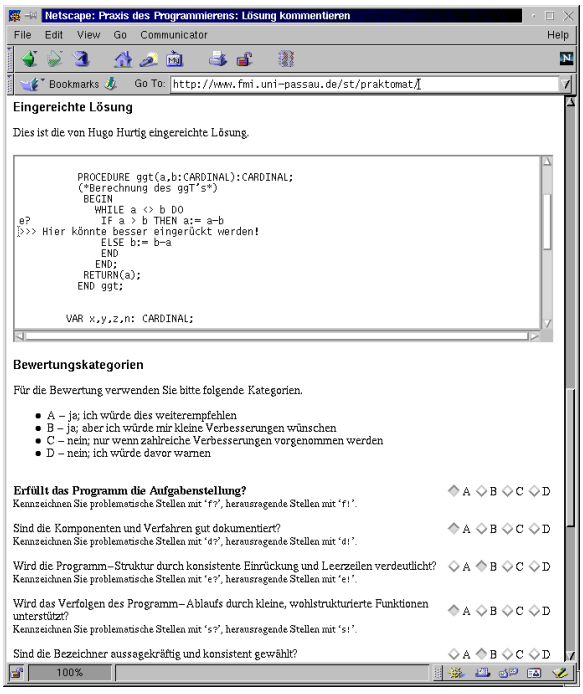

Abbildung 3: Programme kommentieren. Praktomat ermöglicht es Studenten, Lösungen ihrer Kommilitonen zu lesen und zu kommentieren. Dabei können sie den Programmtext mit Kommentaren versehen, aber auch einzelne Eigenschaften des Programms bewerten.

Dokumentation oder die Struktur des Programms einzeln bewerten, um schließlich ihren Gesamteindruck zu hinterlassen.

Im Laufe des Praktikums hat der Großteil der Studenten ein fremdes Programm kommentiert und sich so in die Position des *Lesers* hineinversetzt – eine Erfahrung, von der wir hofften, daß sie auch den eigenen Programmierstil positiv beeinflußt, was denn auch von Studenten in einer Umfrage bestätigt wurde.

Ebenso wichtig wie das Kommentieren fremder Programme war den Studenten natürlich das Erhalten von Kommentaren über die eigenen Programme. Kommentare wurden nach einem *tit for tat*-Prinzip zugeteilt: je mehr und je früher man Kommentare abgegeben hatte, um so größer war die Chance, einen Kommentar zu erhalten. Der gesamte Kommentiervorgang lief ohne Eingriff und Kenntnisnahme durch Mitarbeiter oder sonstige Autoritätspersonen ab – alle Kommentare waren vertraulich und unverbindlich.

Man könnte glauben, ein fremder Kommentar über die eigene Arbeit sei wertvoller als ein eigener Kommentar über eine fremde Arbeit. Die meisten Studenten teilten auch diese Meinung. Einige aber zeigten sich unzufrieden mit den erhaltenen Kommentaren, was auf partielle Probleme mit der Kommentierqualität wie auch den Vorkenntnissen der Kommentatoren deutet. In der begleitenden Vorlesung wollen wir in Zukunft die Bewertung von Programmqualität noch stärker betonen.

6 Betreuen und Bewerten

> Testing is a *destructive* process, even a sadistic process.
>
> — G. J. Myers, *The Art of Software Testing*

Da Praktomat uns große Teile der Qualitätskontrolle abnahm (zeitweise mußte Praktomat 10 Einreicher gleichzeitig bedienen), konnten sich die betreuenden Mitarbeiter voll auf die *Programmierberatung* der Studenten konzentrieren: Jeden Tag wurden zwei Stunden Beratung im Rechnerraum angeboten, darüber hinaus waren wir ganztägig per e-mail erreichbar.

Die Probleme der Studenten unterschieden sich nicht wesentlich von den Problemen in herkömmlichen Programmierpraktika – Programmieren lernen ist hart, ob mit oder ohne Praktomat –, nur daß die Anforderungen an die Funktionalität deutlich höher waren. Praktomatspezifische Probleme gingen schnell vorbei, als die Studenten lernten, die Aufgabenstellungen exakt zu befolgen (insbesondere, was die Syntax der Ein- und Ausgaben anging).

Was die Mitarbeiter aber positiv empfanden, war, daß sie eine reine *Beraterrolle* annehmen konnten. Statt das Programm des Studenten zu kritisieren, hilft man, *die Prüfung zu bestehen*. Diese Prüfung wird durch eine Maschine abgenommen, die unerbittlich, ausdauernd und fair die Programme auseinandernimmt – und so auch all die schlechten Gefühle auf sich zieht, unter denen früher die prüfenden Mitarbeiter zu leiden hatten („Aber mein Programm läuft doch fast!“).

Ein weiterer Vorteil der automatischen Prüfung ist die *Fairneß*: der Anspruch an die Funktionalität des Programms ist für alle Studenten derselbe – und nicht abhängig vom einzelnen Betreuer. Auch die Bewertung der *Verständlichkeit* konnte durch die normierten Kriterien fairer gehalten werden. Schließlich sieht Praktomat vor, das Programm *ohne Kenntnis des Autors* zu bewerten, was die Betreuer unbefangener macht.

Friede, Freude, Eierkuchen: ist das nun das Fazit der Betreuung? Mitnichten. Es gab auch Schattenseiten:

- Die *Realisierung von Praktomat* unter starkem Termin- und Qualitätsdruck war teuer: Praktomat umfaßt heute 10.000 Zeilen Python-Code.
- Das *Prüfen und Testieren* eines Programms per WWW-Textfenster dauert doppelt so lange wie das Korrigieren eines Ausdrucks auf Papier.
- Die *Aufgabenstellungen* waren am teuersten: hier steigt der Aufwand auf das Doppelte bis Dreifache, bedingt durch
  - die erhöhte Präzision,
  - die zahlreichen Varianten,
  - den Entwurf der Testfälle und
  - das Vorab-Implementieren einer *Musterlösung*. (Auch wenn eine Musterlösung in einer sorgfältig gestellten Praktikumsaufgabe eigentlich selbstverständlich sein sollte: Bei Praktomat ist sie zwingend, da nur sie Fehler in den Testfällen aufdeckt.)

Dieser Mehraufwand wird aber ausgeglichen durch die automatische Qualitätskontrolle von Praktomat: gerade die Bewertung der Funktionalität wurde erheblich durch das automatische Testen vereinfacht. Die Realisierung und Pflege des Systems wird sich bei fortschreitendem Einsatz schnell amortisieren. Damit blieb der Gesamtaufwand gleich – bei gesteigertem Lerneffekt für die Studenten.

7 Fazit

The need for reviewing was so obvious to the best programmers that they rarely mentioned it in print, while the worst programmers believed that they were so good that *their* work did not need reviewing.

— Daniel Freedman and Gerald Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews*

Im Urteil der Studenten war die Praktomat-gestützte Lehrveranstaltung ein großer Erfolg. In der offiziellen Lehrevaluation bewerteten die Studenten die Wirksamkeit einzelner Maßnahmen, bezogen auf die Qualität der eigenen Programme:

- 57,7% bestätigten die Wirksamkeit des *automatischen Prüfens* (und 28,8% teilweise)
- 61,5% bestätigten die Wirksamkeit des *Lesens und Kommentierens anderer Programme* (und 19,2% teilweise)
- 63,5% bestätigten die Wirksamkeit des *Kommentierens eigener Programme durch andere* (und 13,5% teilweise)

Diese Zahlen sind ein erstes Meinungsbild; sobald das Praktikum im August 1999 abgeschlossen ist, werden wir exaktere Zahlen im Rahmen einer wissenschaftlichen Auswertung bestimmen können – so z.B. Aussagen über Umfang und Qualität der studentischen Kommentare oder Ähnlichkeitsanalysen über die eingereichten Programme. Der bisherige Eindruck der testierenden Mitarbeiter von der Programmqualität bestätigt das Urteil der Studenten: die Funktionalität ist deutlich besser als in bisherigen Programmierpraktika; die Lesbarkeit steigerte sich spürbar im Laufe des Praktikums.

Fazit: *Best practices* der Industrie wie systematisches Testen oder Software-Reviews lassen sich bereits im ersten großen Programmierpraktikum effizient und qualitätssteigernd einsetzen. Unsere Studenten haben gelernt, sorgfältiger, funktioneller und verständlicher zu programmieren als in herkömmlichen Programmierpraktika – und das bei gleichem Betreuungsaufwand.

Danksagung. Einzelne Praktomat-Konzepte wurden von Versuchen an anderen Hochschulen inspiriert. Dan Hoffman (Victoria) benutzte automatisches Testen im Programmierpraktikum und wies den Autor auf die Problematik präziser Aufgabenstellungen hin. Lutz Prechelt (Karlsruhe) experimentierte im Rahmen des *personal software process* mit gegenseitigem Kommentieren im Hauptstudium und berichtete dem Autor über enthusiastische Studenten. Der Autor dankt beiden für wertvollen Erfahrungsaustausch.

Die Innovationen in Praktomat (so die individuellen Aufgabenstellungen und der Praktomat-Prozeß) profitierten erheblich von den Anregungen und Rückmeldungen des Lehrstuhls: Danke an Jens Krinke, Torsten Robschink, Gregor Snelting und Mirko Streckenbach. Schließlich dankt der Autor noch den 77 Studentinnen und Studenten, deren Verbesserungsvorschläge wesentlich zur Benutzerfreundlichkeit, Stabilität und Funktionalität des Programms beigetragen haben:

Harald Aigner · Cornelia Auer · Ernst Bachmann · Markus Bauer · Martin Bayer · Christian Becker · Hartwig Bentele · Florian Birkenfeld · Ralph Bobrik · Gordon Bolduan · Andreas Braun · Stefan Bredl · Silvia Breu · Genoveva Brunner · Thorsten Buckley · Tobias Bürger · Tobias Eichinger · Tanja Elas · Fou Farah · Katharina Feichtinger · Thomas Fenzl · Thilo Gaul · Tobias Geis · Daniel Gmach · Wolfgang Götz · Martin Grill · Keyuan Gu · Thomas Hackl · Christian Helmbrecht · Stephanie Herrmann · Stella Hoeck · Paul Holleis · Roland Holzhauser · Michael Häusler · Christoph Kamp · Ralf Kern · Manuel Klimek · Ina Klose · Katja Kowalewski · Stephan Kratzer · Jari Krause · Axel Krauth · Richard Kuntschke · Thomas Kühne · Thorsten Lampe · Lorenz Lang · Katrin Limpoeck · Gabriel Linschi · Diana Lucic · Jürgen M. Eckel · Peter M. Fischer · Michael Mandl · Brunner Markus · Jörg Meier · Rank Michael · Gerald Mixa · Stefan Mussner · Claudia Neumar · Bernd Nürnberger · Sebastian Ohlendorf · Christian Pich · Ines Putz · Marc Reich · Stefanie Scherzinger · Bernd Schneider · Martin Schöffler · Wolfgang Stars · Daniel Szentes · Thomas Tatschke · Jürgen Tauschl · Martin Wallraff · Alexander Werni · Frank Westerhausen · Martin Wimmer · Johannes Zapotocz · Cai Ziegler · Thomas Zimmermann.

Praktomat selbst einsetzen

Der Praktomat-Quellcode ist frei verfügbar, so daß Sie Praktomat auch für eigene Lehrveranstaltungen einsetzen können. Technische Voraussetzungen sind ein Python-Interpreter und das DEJAGNU-Testpaket, die ebenfalls frei erhältlich sind.

Praktomat läßt sich leicht an neue Umgebungen anpassen, indem Konfigurationsvariablen gesetzt werden – so etwa der Name der Veranstaltung, der Name der Hochschule, die verwendete Programmiersprache und einige Prozeß-Parameter. Der Praktomat-Code sieht auch tiefere Eingriffe in den Prozeß vor.

Praktomat und weiterführende Literatur finden Sie im WWW unter

http://www.fmi.uni-passau.de/st/praktomat/