Der Grader Praktomat

Joachim Breitner Martin Hecker Gregor Snelting

21. Juli 2016

Kapitel im Buch
Automatisierte Bewertung in der Programmierausbildung

herausgegeben von
Prof. Dr. Oliver J. Bott
Peter Fricke
Dr. Uta Priss
Dr. Michael Striewe

Kapitelautoren: Joachim Breitner, Martin Hecker, Gregor Snelting

Der *Praktomat* (Giffhorn u. a. 2016) ist ein Web-basierter Dienst, auf dem Studenten Programmieraufgaben abrufen und ihre Lösungen einreichen können. Diese werden vollautomatisch durch eine große Zahl von Funktionstests, sowie weitere Tester wie Stilchecker geprüft. Die so erhaltene Rohbewertung der Einreichungen wird durch wissenschaftliche Mitarbeiter finalisiert, wobei zusätzlich softwaretechnische Kriterien herangezogen werden.

Der Praktomat wird seit vielen Jahren an diversen Universitäten in sehr großen Programmierkursen eingesetzt. Er wird zusammen mit dem Plagiatserkenner JPlag betrieben und sorgt nachweislich für eine deutlich verbesserte Programmierkompetenz der Teilnehmer.

1 Geschichte und Überblick

Seine Ursprünge hat der Praktomat an der Universität Passau, als im Jahre 1999 die Abgabe der Programmieraufgaben in der Erstsemesterveranstaltung von Prof. Snelting automatisiert wurde. Die erste Version, von Sneltings erstem Doktoranden Andreas Zeller entwickelt, war noch ein relativ loser Verband von Shell- und Perl-Skripten. Trotz der anfänglichen technologischen Schwächen sorgte das System für eine signifikante Verbesserung des Übungsbetrieb, und führte zu einer nachweislich stark verbesserten Programmierkompetenz der Teilnehmer. Praktomat wurde im Laufe der Jahre von Andreas Zeller, Jens Krinke und anderen weiter ausgebaut (Krinke, Störzer und Zeller 2002).

Im Jahre 2009 stellten Dennis Giffhorn und Daniel Kleinert den Praktomaten für die Erstsemestervorlesung „Programmieren“ von Prof. Snelting, inzwischen am KIT, auf die heutige technische Basis: Der Praktomat ist in Python implementiert und basiert auf dem Web-Framework Django. Seit dem wurde der Praktomat von weiteren Mitarbeitern von Prof. Snelting weiterentwickelt, insbesondere Martin Hecker und Joachim Breitner.

Der Praktomat wird heute in diversen Veranstaltungen verschiedener Universitäten verwendet. Insbesondere wird am KIT die Veranstaltung „Programmieren“ für Informatik-Erstsemester mit jährlich ca. 800 – 900 Teilnehmern mit dem Praktomaten abgewickelt.

Die Programmierkompetenz der Teilnehmer verbessert sich nachweislich vor allem durch folgende Faktoren:

- Die Aufgabenstellungen müssen – da die eingereichten Programme vollautomatisch getestet werden – sehr genau spezifiziert sein, bis in kleinste Details der Ein-/ Ausgabe. Dies führt zu einer intensiven Beschäftigung mit der Aufgabe und zu einer großen Sorgfalt der Teilnehmer.

- Das automatische Testen anhand ca. 50 – 200 geheimen Testfällen pro Aufgabe, in Kombination mit automatischem Check des Java Style Guides (`checkstyle`, *Checkstyle* 2016), führt zu einer belastbaren, *objektiven*, präzisen, und anonymisierten Bewertung der Programmierkompetenz.

- Die abschließende Bewertung softwaretechnischer Kriterien durch wissenschaftliche Mitarbeiter erzwingt schon im ersten Semester eine gute (objektorientierte) Programmarchitektur.

- Praktomat wird immer in Kombination mit dem am KIT entwickelten Plagiatsprüfer JPlag eingesetzt (der laut eines Vergleichstests der international beste Java-Plagiatserkenner ist, siehe Weber-Wulff 2012); so werden Teilnehmer gezwungen, wirklich selbst zu programmieren.

Diese Vorteile erlaubten es, auf eine Abschlussklausur in der Vorlesung „Programmieren“ zu verzichten und stattdessen zwei anspruchsvolle Abschlussaufgaben (mit je ca. 500 – 1500 LOC) als Prüfungsleistungen festzulegen; dies ist didaktisch als wesentlich effektiver anzusehen („*learning by doing*“) und verstärkt den kompetenzfördernden Effekt von Praktomat.

2 Abläufe und Besonderheiten

Mit dem Praktomaten interagiert man meist in einer von vier Rollen:

- als der Administrator des Systems, auf dem der Praktomat läuft;

- als Übungsleiter, der die Aufgaben erstellt und die letzendliche Verantwortung für die Benotung der Studenten trägt;

- natürlich als Student, der seine Lösung einreicht;

- als (meist studentischer) Tutor, der die Lösungen sichtet, kommentiert und bewertet.

Bei den großen Programmiervorlesungen sind diese Rollen disjunkt, während bei kleineren Vorlesungen der Übungsleiter oft auch die Rolle des Tutors übernimmt.

Im Folgenden beschreiben wir die Abläufe und Besonderheiten des Praktomaten aus der jeweiligen Rollensicht.

2.1 ... aus Sicht des Administrators

Der Praktomat ist, technisch gesehen, eine weitgehend herkömmliche Django-Anwendung und läuft auf einem üblichen Linux-basierten System. In Karlsruhe wird Debian `stable` eingesetzt. Der Quellcode, der unter den Bedingungen der GPL-Lizenz Version 2 verteilt wird, ist über Github[1] zu beziehen; die enthaltene README-Datei beschreibt den Installationsprozess.

Es wird eine Datenbank vorausgesetzt. Prinzipiell unterstützt der Praktomat – dank Django – dabei eine Reihe von verschiedenen Systemen. In der Praxis getestet ist dabei allerdings nur Postgresql und sqlite, wobei letzters aus Performance-Gründen nur für Test- und Entwicklungsinstanzen geeignet ist. Weiter wird als Webserver apache eingesetzt.

Für die Verwaltung der Benutzer (Übungsleiter, Tutoren, Studenten) kann die Django-eigene Benutzerverwaltung verwendet werden, mit der üblichen Registrierung per Bestätigungs-e-Mail und Authentifizierung per Benutzername und Passwort. Komfortabler ist allerdings die Integration in ein Universitäts-weites Single-Sign-On-System. Hier unterstützt der Praktomat Shibboleth, mit Hilfe des Apache-Moduls `mod-shib2`.

Aus Sicht des Praktomaten gibt es nur eine Vorlesung; er ist nicht mandatenfähig. Um trotzdem auf einem Server die Praktomat-Instanzen für verschiedene Vorlesungen laufen zu lassen, können mehrere Praktomatinstanzen, jeweils mit eigener Datenbank, in verschiedenen Verzeichnissen installiert werden und so völlig unabhängig voneinander betrieben werden.

Für das Testen des nicht vertrauenswürdigem studentischem Code legt der Administrator entweder einen dedizierten Benutzer oder – besser – einen dedizierten Container in der Container-Lösung `docker` an. Mehr zur Sicherheitsarchitektur in Abschnitt 3.

Der Administrator legt fest, wie viele Prozesse pro Praktomat-Instanz die Benutzeranfragen bearbeiten. Da Einreichungen synchron getestet werden, also die HTTP-Abfrage entsprechend lange läuft, sorgen zu wenige Prozesse dazu, dass der Praktomat nicht erreichbar ist, wenn alle Prozesse ausgelastet sind; daher sollte diese Zahl nicht zu klein gewählt werden.

Eine Skalierung in die Breite, also über mehrere Server, wird nicht unterstützt. Da sowohl das Webinterface als auch die Prüfung der Abgaben auf dem gleichen System läuft, sollte dieses Adäquat ausgestattet sein.

In Karlsruhe kommt bei 10 Prozessen pro Instanz ein virtuelles System mit 4GB RAM zum Einsatz.

1 `https://github.com/KITPraktomatTeam/Praktomat`

Sprache	Kompilation	Unit-Tests	Stilprüfung
Java	✓	✓	✓
C/C++/Fortran	✓		
Haskell	✓	✓	
R	✓		
Isabelle	✓		

Tabelle 1: Programmiersprachen-spezifische Features

2.2 ... aus Sicht des Übungsleiters

Der Übungsleiter, auch *trainer* im Praktomat-Jargon, ist der für den Übungsbetrieb verantwortliche Dozent oder Mitarbeiter. Er ist der einzige, der mit dem Praktomaten auch über die sogenannte „Admin-Oberfläche“ interagiert.

Dort legt er Tutorien (Übungsgruppen) an, ernennt Benutzer zu Tutoren, und weist den Studenten diese Gruppen zu.

Weiter legt er hier die Aufgaben (*tasks* im Praktomat-Jargon) an. Diese bestehen aus einem Titel, einer Aufgabenbeschreibung aus formatierbarem Text, eventuellen Anhängen und einer Reihe von *Checkern*. Diese Checker sind das Herz des Praktomaten, und stellen abstrakt einen Schritt in der automatischen Prüfung der studentischen Abgabe dar. Manche Checker stellen Funktionalität bezüglich einer bestimmten Programmiersprache bereit (siehe Tabelle 1), andere sind generisch. Die wichtigsten Checker sind:

- Ein Java-Checker, der Java-Code in der studentischen Abgabe sucht und kompiliert. Analoge Checker existieren für C, C++, Fortran, Haskell und Isabelle.
- Stil-Checker, die mit Hilfe von `checkstyle` gewisse, vom Übungsleiter vorgegebene Stilvorgaben prüfen.
- Unittest-Checker, die vom Übungsleiter vorgegebene JUnit- oder DejaGnu-Tests gegen den studentischen Code laufen lassen. Haskell-Abgaben werden durch HUnit- und QuickCheck-Tests überprüft.
- Für die Sprache R existiert ein spezieller Checker, der ein bei der Ausführung des R-Codes erzeugtes PDF-Dokument abspeichert, damit es später bei der Begutachtung der Einreichungen betrachtet werden kann.
- Ein generischer Skript-Checker, der es den Übungsleiter erlaubt, beliebigen Code (etwa Python- oder Bash-Skripte) im Kontext der studentischen Abgabe laufen zu lassen. Dieser Checker bietet dem Übungsleiter maximale

Freiheiten, und einige Übungsleiter nutzen diesen, um mit selbst entwickelten Frameworks aufwendige Akzeptanz-Tests gegen das eingereichte Programm laufen zu lassen.

Jeder Checker produziert einen Ausgabetext und einen Status, der den Erfolg des Checkers angibt.

Der Ausgabetext kann dabei entweder reiner Text sein (etwa die Status-Meldungen des Compilers) oder HTML-Fragmente. Letzteres erlaubt es insbesondere dem Skript eines Skript-Checkers, ansprechende und interaktive dynamische Ausgaben zu erstellen.

Der Übungsleiter markiert jeden Checker als zwingend nötig oder optional sowie als privat oder öffentlich. Meldet ein zwingend nötiger Checker einen Fehlschlag, so werden keine weiteren Checker ausgeführt und die Einreichung wird gar nicht erst angenommen. So kann erreicht werden, das nicht kompilierender Code nicht bewertet wird. Die Ergebnisse von privaten Checkern werden dem Studenten nicht angezeigt, sondern sind nur dem Übungsleiter und dem Tutor sichtbar.

Über die Funktion, Testabgaben hochzuladen, kann der Übungsleiter die Konfiguration der Aufgabe überprüfen.

Die Bewertungsskala für die Aufgabe (z.B. bestanden/nicht bestanden, Schulnoten, Zahlenbereiche etc.) ist auch konfigurierbar. Auch mehrere Skalen (z.B. Funktionalität, Stil) können angelegt werden.

Zuletzt legt der Übungsleiter fest, ab wann die Aufgabe für die Studenten sichtbar ist und bis wann Einreichungen angenommen werden sollen.

Nach dem Ende der Einreichungsfrist stößt der Übungsleiter einmal die erneute Prüfung aller finalen Abgaben an, bevor die Tutoren die Einreichungen zu sehen bekommen. So wird sichergestellt, dass auch bei späteren Verbesserungen der Checker alle Bewertungen mit den selben, finalen Versionen der Checker bewertet werden.

Auch kann der Übungsleiter nun mit einem Klick alle Abgaben von der Plagiatserkennungs-Software `jPlag` (*jPlag* 2016) prüfen lassen. Der dabei produzierte Bericht listet auf, welche Abgaben sich verdächtig stark ähneln. Dabei lässt sich `jPlag` bei unterstützten Programmiersprachen (Java, C#, C, C++, Scheme) nicht von naiven Verschleierungstechniken wie dem Umbenennen von Variablen täuschen.

Nachdem die Tutoren die Aufgaben bewertet haben, und sofern die Konfiguration vorgibt, dass die Tutoren das nicht selber machen können, gibt der Übungsleiter die Bewertungen frei. Die Studenten erhalten eine Nachricht per e-Mail und können nun die Bewertung sehen.

Zum Abschluss des Semesters kann der Übungsleiter aus den Bewertungen der einzelnen Übungsblätter die Gesamtnote berechnen lassen. Das Schema, nachdem die Einzelbewertungen verrechnet werden, gibt er dabei als JavaScript-Funktion an; so sind hier große Freiheiten möglich.

2.3 ... aus Sicht des Studenten

Für die Studenten beginnt die Interaktion mit dem Praktomaten dann, wenn die vom Übungsleiter erstellten Aufgaben veröffentlicht werden. Von nun an können sie ihre Lösungen hochladen. Bei Abgaben, die aus mehreren Dateien bestehen, können diese entweder einzeln hochgeladen werden, oder als Archivdatei, die vom Praktomaten dann automatisch entpackt wird.

Die vom Übungsleiter wie oben beschriebenen Checker werden unnmittelbar auf die Lösung des Studenten angewandt. Dabei werden dem Studenten die Ausgaben der als öffentlich markierten Checker angezeigt, so dass er seine Abgabe entsprechen verbessern und erneut einreichen kann. Schlägt dabei ein zwingend nötiger Checker fehl, so wird die Einreichung abgelehnt, ansonsten angenommen. Der Student hat eine Übersicht über seine bisherige Einreichungen und sieht, welche angenommen wurden und welche derzeit die *finale* Abgabe ist. In der Regel ist das die jüngste, bei der keine zwingenden Tests fehlgeschlagen sind, er kann aber auch manuell eine ältere als final markieren.

Bis zur Deadline kann der Student beliebig oft eine neue Lösung eintragen.

Sobald sein Tutor die Abgabe wie im nächsten Abschnitt beschrieben bewertet hat bekommt der Student per e-Mail eine Benachrichtigung und kann die Bewertung lesen.

2.4 ... aus Sicht der Tutors

Die Aufgabe eines Tutors ist es, die Abgaben der Studenten in seiner Übungsgruppe zu begutachten. Dabei können einer Übungsgruppe auch mehrere Tutoren zugewiesen werden, die sich die Arbeit teilen.

Der Tutor sieht die Ausgabe aller Checker, auch der als privat markierten, sowie den Quellcode des Studenten, mit ensprechendem Syntax-Highlighting (Abbildung 1). Die Quelldateien darf er bearbeiten, etwa um bestimme Code-Zeilen zu kommentieren oder Verbesserungsvorschläge zu machen. Dem Studenten werden diese Änderung mit entsprechend deutlicher farblicher Hervorhebung als *diff* angezeigt (Abbildung 2).

Zusätzlich hat der Tutor die Möglichkeit, die Begutachtung (*attestation* im Praktomat-Jargon) mit einem für den Studenten sichtbaren Kommentar sowie mit einer privaten, nur für den Übungsleiter sichtbaren, Bemerkung versehen. Letzte-

Abbildung 1: Studentische Abgabe mit Testresultaten

Programmieren Abschlussaufgaben WS 2015/2016

Welcome, Joachim Breitner / View Account / Log-out
Rating overview / Admin Panel

Home > Abschlussaufgabe 2 > My solutions > Solution 3486 > Attestation 796

Attestation: Abschlussaufgabe 2

by Some Tutor for Some Student

Private Comment

Nach Korrektur noch funktionale Punkte anpassen.

Comment

OO-Modellierung: CommandLine sollte normale Instanzmethoden besitzen, weil der Parameter Pitch überall verwendet wird. Spiellogik sollte von Kommandozeilenein- und ausgabe getrennt werden
Komplexität: CommandLine:pitch , Pitch::check* sind zu lange, obwohl die Methoden unterteilbar wäre. Schachtelung des Kontrollflusses teilweise zu tief.
Kapselung: Prüfung und Umrechnung von Koordinaten sollte im Spielfeld stattfinden, weil nur dort Dimensionen bekannt sein sollten.
Programmierstil: Magische Zahlen in CommandLine.

Nach anpassen der Lösung sodass ein Unentschieden korrekt erkannt wird laufen einige Tests mehr durch. Es gibt also 1.5P mehr auf die Funktionalität.

Ratings

Objektorientierte Modellierung: 1.5

Kapselung: 0.5

Verständlichkeit und Komplexität: 0.5

Programmierstil: 0.5

Kommentarpraxis und Javadoc: 1.0

Funktionalität: 11.5

Final grade: 15.5

Annotated Files

Download Solution

Figure.java | CommandLine.java* | Pitch.java | SyntaxException.java | PitchTorus.java | Main.java

```
package edu.kit.informatik;

/**
```

```
        String[] params = checkArguments(args[1], 1);
        int id = isPositiveInteger(params[0]);

        /* we don't need to look for 0 < id since it's already done. */
-       if (id > 15) {
+       if (id > 15) { // Selbsterklärende Konstanten wären hilfreich!
            throw new SyntaxException("a valid figure number is a number between 0 and 15.");
        }
        if (pitch.getChosenFigure() != null) {
            throw new SyntaxException("figure '" + pitch.getChosenFigure().getNumber()
```

Abbildung 2: Begutachtung mit Kommentaren im Quelltext

re kann genutzt werden, um diesen auf mögliche Betrugsfälle hinzuweisen. Die Bewertung oder – bei mehreren Skalen – Bewertungen legt der Tutor ebenfalls fest.

Sobald der Tutor zufrieden ist, markiert er die Bewertung als final; finale Bewertungen werden je nach Konfiguration entweder vom Übungsleiter oder vom Tutor selbst veröffentlicht. Nach der Veröffentlichung kann der Student den Kommentar, die Bewertung und die Änderungen am Quellcode einsehen. Der Tutor kann nun keine Änderungen mehr vornehmen, eine bereits dem Studenten veröffentlichte Begutachtung kann nur vom Übungsleiter zurückgezogen werden.

Selbstverständlich sehen Studenten nur ihre eigenen Abgaben und Bewertungen, und Tutoren nur die aus der ihnen zugewiesenen Übungsgruppe.

3 Sicherheitskonzept

Der Praktomat führt notwendigerweise von den Studenten eingereichten und damit nicht vertrauenswürdigen Programmcode aus. Es muss also dafür gesorgt sein, dass auch ein böswilliger Student nicht Daten lesen kann, die er nicht lesen darf, und erst recht nicht die Kontrolle über den Server übernehmen kann.

Der Haupteinsatz von Praktomat ist in Vorlesungen, in denen Java gelehrt wird. Hier wird der Java Security Manager so konfiguriert, dass nur Dateien im aktuellen Verzeichnis gelesen und geschrieben werden dürfen, und kein Netzwerkzugriff möglich ist.

Inzwischen wird der Praktomat allerdings auch mit anderen Programmiersprachen (z.B. R, Isabelle) verwendet, die keine solchen eingebauten Sicherheitsmechanismen mitbringen. Hier wird zusätzlich die Ausführung jedes potentiell gefährlichen Befehls, was neben der Ausführung auch die Kompilation des Benutzercodes einschließt, in einen für diesen Zweck gestarteten Docker-Container verlegt. Dieses „virtuelle“ System hat sein eigenes, nur schreibbares Dateisystem, sein eigenes temporäres Verzeichnis und kein Netzwerkzugriff. Lediglich das das Verzeichnis mit den eingereichten Dateien wird per *bind-mount* in den Container hineingereicht.

Container bieten hier die nötige Kapselung und zusätzlich eine komfortable Beschränkung des dem Programm zur Verfügung stehenden Hauptspeicher und der Laufzeit. Diese Funktionalität ist als `safe-docker` (Breitner 2016) auch unabhängig vom Praktomaten verfügbar.

Tabelle 2: Einsatz von Praktomat am KIT seit 2011

Semester	Studenten	Tutoren	Tasks	Abgaben	$\sum$ LOC	$\oslash$ LOC
Algorithmen 1 (Java)						
SS 2014	267	26	5	391	101 417	259
Computational Risk and Asset Management (R, Python)						
WS 2014	34	3	12	256	55 427	216
WS 2015	21	2	13	173	34 949	202
Programmieren (Java)						
SS 2011	90	4	11	380	108 784	286
WS 2011	848	29	10	3 455	5 101 898	1476
SS 2012	203	8	12	807	235 509	291
WS 2012	864	37	11	4 959	1 781 767	359
SS 2013	267	7	7	378	98 041	259
WS 2013	724	27	9	3 623	1 198 904	330
SS 2014	238	5	9	798	204 685	256
WS 2014	798	30	20	7 899	1 377 868	174
SS 2015	212	10	12	1 212	236 248	194
WS 2015	933	29	22	12 210	1 495 461	122
SS 2016	192	8	9	803	200 314	249
Programmieren: Abschlussaufgaben (Java)						
SS 2011	105	6	2	84	152 027	1809
WS 2011	855	30	2	972	1 538 585	1582
SS 2012	147	4	2	176	188 556	1071
WS 2012	590	23	2	922	1 076 255	1167
SS 2013	187	7	2	154	96 171	624
WS 2013	391	20	2	596	673 590	1130
SS 2014	126	11	2	187	167 701	896
WS 2014	397	24	2	703	949 137	1350
SS 2015	133	8	2	222	189 844	855
WS 2015	453	17	2	797	1 053 362	1321
Theorembeweiser: Anwendungen in der Sprachtechnologie (Isabelle)						
SS 2013	29	1	7	51	20 788	407
SS 2016	18	2	6	53	15 472	291

LOC: lines of code der Abgaben, $\oslash$ pro Task, beinhaltet Leer-, Kommentarzeilen.
Stand der Zahlen für SS 2016: 10.06.2016.

4 Einsatz

Tabelle 2 zeigt den Umfang des Praktomat-Einsatzes am KIT in den letzten 5 Jahren. Neben „Programmieren“ wurde Praktomat an der Informatik-Fakultät eingesetzt in „Algorithmen 1“ (Java) sowie im Praktikum „Theorembeweiser: Anwendung in der Sprachtechnologie“. Im letzteren werden keine Programme eingereicht, sondern mit dem Theorembeweiser Isabelle erstellte formale Beweise, deren Korrektheit dann server-seitig geprüft wurde. Die Formel-lastigen Beweise reizten die Unicode- und Syntax-Highlighting-Fähigkeiten des Praktomaten aus.

Auch jenseits der Informatik kommt der Praktomat zum Einsatz: Die Fakultät für Wirtschaftswissenschaften des KIT setzt den Praktomaten in einer Vorlesung zum Risikomanagement ein.

Außerhalb von Karlsruhe sind uns Praktomat-Instanzen an der HS Emden, der HFT Leipzig, dem University College London, der HS Regensburg und der TU Dresden bekannt.

An der HS Ostfalia enstand im Rahmen des eCult-Projektes eine Integration des Praktomaten in das LMS LON-CAPA (siehe Kapitel **??**), bei der der Praktomat im Hintergrund über eine REST-Schnittstelle angesprochen wird, wärend die Studenten lediglich mit LON-CAPA interagieren (Kruse und Jensen 2013; Maier und Rod 2015).

5 Erfahrungen

Allein am KIT haben seit 2008 viele tausend Informatikstudenten ihre Programmieraufgaben über den Praktomaten eingereicht. Zum Schluss dieser Beschreibung von Praktomat möchten wir deshalb einige strategische Auswirkungen des Praktomat-Einsatzes aufzeigen, die nachgerade „politischer“ Natur sind.

Praktomat und PSE. Die Veranstaltung „Praxis der Softwareentwicklung“ (PSE) wurde 2009 an der Informatik-Fakultät des KIT eingeführt mit dem Ziel, Drittsemestern Kompetenz in professioneller Softwareentwicklung zu vermitteln. PSE ist charakterisiert durch: Arbeit in Teams von 5 Teilnehmern; Entwicklung eines mittelgroßen Projektes durch alle Phasen vom Pflichtenheft bis zur Abschlusspräsentation; objektorientierter Entwurf mit UML; integrierte Qualitätssicherung. Alle Lehrstühle der Fakultät stellen Aufgaben, dabei ist das Prozessmodell vorgegeben, inhaltlich sind die Lehrstühle aber frei. Die meisten Lehrstühle wählen Themen aus ihrem Forschungsbereich (Stichwort „forschungsorientierte Lehre“).

Die Vorgabe für die Systemgröße ist ca. 100 Klassen, ca. 10000 LOC (meist Java, aber auch C#; oft Android-Apps). PSE hat 8 LP, also nominell insgesamt

$5 \times 8 \times 30 = 1200$ Arbeitsstunden – was, verglichen mit typischen Industriekennzahlen, sehr wenig ist für ein Produkt von 10000 LOC. Alle Teams erzielen zumeist glänzende Ergebnisse mit Industriequalität (auch in der GUI), und preisen ihre Projekte in ihrem CV, im Internet und anderswo an.

Möglich ist das nur durch die mittels Praktomat erworbenen Vorkenntnisse in Programmieren. Frühere PSE-ähnliche Veranstaltungen des Lehrstuhl Snelting hatten klassische „manuelle“ Programmierkurse als Vorstufe, so dass bei weitem nicht die Größe und Professionalität von PSE-Projekten erreicht wurde. Entscheidend ist die stringente Qualitätssicherung durch das automatische Testen. Der Lehrstuhl Snelting erhielt für die Einführung von PSE den Fakultätslehrpreis.

Systemakkreditierung. Ein interessanter, „politischer“ Nebeneffekt von Praktomat ergab sich 2014 im Zuge der Systemakkreditierung des KIT-Informatik-Studiengangs. Die Frauenbeauftragte hatte festgestellt, dass sich die durchschnittlichen Bachelor-Abschlussnoten von Männern und Frauen um ca. 0,25 Notenpunkte unterscheiden, und dafür eine (bewusste oder unbewusste) Diskriminierung von Studentinnen durch die Dozenten verantwortlich gemacht. Da „Programmieren“ die größte Veranstaltung der KIT Informatik ist, wurden Daten aus Praktomat zur Überprüfung dieser Behauptung analysiert. In der Tat schnitten in den Jahre 2008 – 2010 Frauen durchschnittlich ca. 0,3 Notenpunkte schlechter in „Programmieren“ ab. In diesen Jahren wurden Aufgaben strikt anonymisiert bewertet. Es war den Bewertern definitiv unmöglich, das Geschlecht (oder andere Eigenschaften) der Teilnehmer zu erkennen: Praktomat zeigte keinerlei Informationen zu Einreichern an, und Einreichungen, die Hinweise auf den Autor im Quelltext enthielten, wurden zurückgewiesen.

Ab 2011 konnten die Bewerter (auf Wunsch des damaligen Dozenten) die Namen der Einreicher sehen. Daraufhin glichen sich die Durchschnittsnoten von Männern und Frauen in „Programmieren“ an. Da der Schwierigkeitsgrad der Aufgaben und die Bewertungskriterien gleich blieben (insbesondere wird weiterhin vollautomatisch getestet) war gezeigt, dass es entweder ab 2011 eine signifikante Veränderung in der Erstsemesterpopulation gab, oder dass die Betreuer Frauen (bewusst oder unbewusst) *besser* bewerten als Männer. Die Frauenbeauftragte zog daraufhin ihre Behauptung zurück.

Plagiatsversuche. Ein anderes, offenes Problem sind Betrugsversuche. Zwar erkennt JPlag zuverlässig Plagiate, insbesondere Kopien und Varianten von eingereichten Programmen (weil z.B. ein Teilnehmer das Programm abgeschrieben hat); auch geänderte Variablennamen oder THEN-ELSE Vertauschungen nützen nichts. Wenn ein Teilnehmer sich jedoch das Programm extern erstellen lässt, und

es nur einmal eingereicht wird, kann JPlag das nicht erkennen. Nachdem mehrere derartige Fälle aufgeflogen sind – weil externe „Nachhilfelehrer“ Praktomat-spezifische Programmier-Dienste gegen Bezahlung im Internet anboten – sollen nun die Abschlussaufgaben in „Programmieren“ durch eine Klausur ergänzt werden. Da aber weiterhin zwei Abschlussaufgaben bearbeitet werden müssen, wird dies den kompetenzsteigernden Effekt des Praktomaten nicht beeinträchtigen.

Abschließend lässt sich feststellen: Die Entwicklung und der Einsatz von Praktomat waren und sind ein überwältigender Erfolg. Die Programmierkompetenz wird durch Praktomat stark verbessert – wir arbeiten z.Zt. an einer statistisch belastbaren Studie, die auf Praktomat-Rohdaten beruht, wobei PSE- und Praktomat-Noten korreliert werden. Auch ohne belastbare Statistik zeigen weiterführende Software-Projekte seit Jahren glänzende Ergebnisse, die nach Überzeugung aller Beteiligten ohne Praktomat nicht möglich wären.

Literaturverzeichnis

[Bre16] Joachim Breitner. *safe-docker*. 2016. URL: `https://github.com/nomeata/safe-docker` (besucht am 09. 06. 2016).

[Che] *Checkstyle*. 2016. URL: `http://checkstyle.sourceforge.net/` (besucht am 09. 06. 2016).

[Gif+16] Dennis Giffhorn u. a. *Praktomat*. 2016. URL: `http://pp.info.uni-karlsruhe.de/projects/praktomat/praktomat.php` (besucht am 09. 06. 2016).

[Jpl] *jPlag*. 2016. URL: `https://jplag.ipd.kit.edu/` (besucht am 09. 06. 2016).

[KJ13] Marcel Kruse und Nils Jensen. “Automatische Bewertung von Datenbankaufgaben unter Verwendung von LON-CAPA und Praktomat”. In: *ABP*. Bd. 1067. CEUR Workshop Proceedings. CEUR-WS.org, 2013.

[KSZ02] Jens Krinke, Maximilian Störzer und Andreas Zeller. “Web-basierte Programmierpraktika mit Praktomat”. In: *Softwaretechnik-Trends* 22.3 (Okt. 2002).

[MR15] Sebastian Maier und Oliver Rod. “Umsetzung eines Laborversuches auf LON-CAPA mit automatischer Bewertung von ABEL-Programmieraufgaben”. In: *ABP*. Bd. 1496. CEUR Workshop Proceedings. CEUR-WS.org, 2015.

[WW12] Debora Weber-Wulff. *Collusion Detection System Test Report 2012*. 2012. URL: http://plagiat.htw-berlin.de/collusion-test-2012/ (besucht am 09. 06. 2016).