

aus

Oliver J. Bott, Peter Fricke, Uta Priss, Michael Striewe (Hrsg.)

Automatisierte Bewertung in der Programmierausbildung

Digitale Medien in der Hochschullehre Band 6

2017, 420 Seiten, br., 42,90 €, ISBN 978-3-8309-3606-0



Waxmann Verlag GmbH

www.waxmann.com info@waxmann.com

10 Der Grader Praktomat

Joachim Breitner, Martin Hecker und Gregor Snelting

Zusammenfassung

Der Praktomat [Gif+16] ist ein webbasierter Dienst, auf dem Studierende 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 Plagiatserkennner JPlag betrieben und sorgt nachweislich für eine deutlich verbesserte Programmierkompetenz der Teilnehmer.

10.1 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 Übungsbetriebs 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 [KSZ02].

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 Informatikerstsemester 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 geheimer Testfälle pro Aufgabe, in Kombination mit automatischem Check des Java Style Guides (`checkstyle`, [Che]), 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-Plagiaterkennung ist, siehe [WW12]); 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.

10.2 Abläufe und Besonderheiten

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

- als Administrator des Systems, auf dem der Praktomat läuft;
- als Übungsleiter, der die Aufgaben erstellt und die letztendliche 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.

10.2.1 ... 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 Github¹ 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 letzteres 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 universitätsweites 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 Praktomat-Instanzen, jeweils mit eigener Datenbank, in verschiedenen Verzeichnissen installiert werden und so völlig unabhängig voneinander betrieben werden.

Für das Testen des nicht vertrauenswürdigen studentischen Codes legt der Administrator entweder einen dedizierten Benutzer oder – besser – einen dedizierten Container in der Containerlösung `docker` an. Mehr zur Sicherheitsarchitektur in Abschnitt 10.3.

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 dafür, dass

¹ <https://github.com/KITPraktomatTeam/Praktomat>

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.

10.2.2 ... 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 10.1), 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.

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

Tabelle 10.1: Programmiersprachenspezifische Features

- Unit-Test-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 dem 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 Akzeptanztests gegen das eingereichte Programm laufen zu lassen.

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

Der Ausgabertext kann dabei entweder reiner Text sein (etwa die Statusmeldungen 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, dass 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 denselben, finalen Versionen der Checker bewertet werden.

Auch kann der Übungsleiter nun mit einem Klick alle Abgaben von der Plagiatserkennungssoftware `jPlag [Jpl]` 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 Verschleiertechniken 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.

10.2.3 ... 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 unmittelbar auf die Lösung des Studenten angewandt. Dabei werden dem Studenten die Ausgaben der als öffentlich markierten Checker angezeigt, so dass er seine Abgabe entsprechend 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 bisherigen 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, aber er kann 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.

Programmieren Abschlussaufgaben WS 2015/2016

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

Home > Abschlussaufgabe 2 > My solutions > Solution 3

Abschlussaufgabe 2

All required tests have been passed. Nevertheless there is at least one warning
This is your current final solution.

Results

- ▶ **Checkstyle: Required Rules** : passed
- ▶ **Checkstyle: Optional Rules** : passed
- ▶ **Checkstyle: Secret Tests** ⚠ : failed (but not required)

- ▼ **Basic Tests** ⚠ : failed (but not required)

+ Bag als erster befehl (0.5p)
Success

+ Einfaches select/place im standard spielfeld. (0.5p)
Failure: Found 2 serious issues

- Select/place und colprint im standard spielfeld. (0.5p)

Description:

Select/Place und Colprint im Standard Spielfeld.

Program execution:

Terminal

```

> java edu.kit.informatik.Main standard
Active normalisations: lowercase
select 11
ok
place 1;4
Error: Unknown Error
Expected call to Terminal.println(), but got error message
Expected call to Terminal.println(), but got call to Terminal.readLine()
select 10
ok
place 2;4
Error: Unknown Error
Expected call to Terminal.println(), but got error message
Expected call to Terminal.println(), but got call to Terminal.readLine()
colprint 4
# 11 10 # # #
quit
Successfully detected program exit.
>
Time taken: 0.50s

```

System.in
 System.out
 System.err

Failure: Found 4 serious issues

+ Beispiel aus dem Übungsblatt (0.5p)
Failure: Found 9 serious issues

3 successes
7 failures

- ▶ **Advanced Tests** ⚠ : failed (but not required)
- ▶ **Error Handling** ⚠ : failed (but not required)

Files

Ball.java
Main.java
StandardPitch.java
TorusPitch.java

```

1 package edu.kit.informatik;
2
3 import java.util.Comparator;

```

Download

Abbildung 10.1: Studentische Abgabe mit Testresultaten

Programmieren Abschlusssaufgaben WS 2015/2016

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

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

Attestation: Abschlusssaufgabe 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 Kommandozeilein- 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

```

1 package edu.kit.informatik;
2
3 /**
4  *
5  */
6
7 import java.util.*;
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Abbildung 10.2: Begutachtung mit Kommentaren im Quelltext

10.2.4 ... aus Sicht des 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 entsprechendem Syntax-Highlighting (Abbildung 10.1). Die Quelldateien darf er bearbeiten, etwa um bestimmte Codezeilen zu kommentieren oder Verbesserungsvorschläge zu machen. Dem Studenten werden diese Änderung mit entsprechend deutlicher farblicher Hervorhebung als *diff* angezeigt (Abbildung 10.2).

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. Letztere 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 Studierende nur ihre eigenen Abgaben und Bewertungen, und Tutoren nur die aus der ihnen zugewiesenen Übungsgruppe.

10.3 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 potenziell 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 keinen Netzwerkzugriff. Lediglich 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 Hauptspeichers und der Laufzeit. Diese Funktionalität ist als *safe-docker* [Bre16] auch unabhängig vom Praktomaten verfügbar.

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.

Tabelle 10.2: Einsatz von Praktomat am KIT seit 2011

10.4 Einsatz

Tabelle 10.2 zeigt den Umfang des Praktomat-Einsatzes am KIT in den letzten 5 Jahren. Neben „Programmieren“ wurde Praktomat an der Informatikfakultät 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 serverseitig geprüft wurde. Die formellastigen 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 entstand im Rahmen des eCult-Projektes eine Integration des Praktomaten in das LMS LON-CAPA (siehe Kapitel 19), bei der der Praktomat im Hintergrund über eine REST-Schnittstelle angesprochen wird, während die Studierenden lediglich mit LON-CAPA interagieren [KJ13; MR15].

10.5 Erfahrungen

Allein am KIT haben seit 2008 viele tausend Informatikstudierende 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 Informatikfakultät 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 Industrie-kennzahlen, sehr wenig ist für ein Produkt von 10000 LOC. Alle Teams erzielten 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 des Einsatzes von Praktomat ergab sich 2014 im Zuge der Systemakkreditierung des KIT-Informatikstudiengangs. Die Frauenbeauftragte hatte festgestellt, dass sich die durchschnittlichen Bachelorabschlussnoten 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 aufgefliegen sind – weil externe „Nachhilfelehrer“ Praktomat-spezifische Programmierdienste 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 Softwareprojekte seit Jahren glänzende Ergebnisse, die nach Überzeugung aller Beteiligten ohne Praktomat nicht möglich wären.

Literatur für dieses Kapitel

- [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: *Workshop „Automatische Bewertung von Programmieraufgaben“ (ABP 2013)*. 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 (2002), S. 51–53.
- [MR15] Sebastian Maier und Oliver Rod. „Umsetzung eines Laborversuches auf LON-CAPA mit automatischer Bewertung von ABEL-Programmieraufgaben“. In: *Workshop „Automatische Bewertung von Program-*

mieraufgaben“ (ABP 2015). 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).