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

15 Der Grader PABS

Lukas Iffländer und Alexander Dallmann

Zusammenfassung
Der modulare Grader PABS unterstützt die Programmiersprache Java sowie andere, auf der Java Virtual Machine (JVM) basierende, Sprachen und wird an der Universität Würzburg entwickelt und eingesetzt. Er dient zur Unterstützung von Vorlesungen und zur Durchführungen von Prüfungen in Praktika im Rahmen der Ausbildung der Würzburger Informatikstudenten und in angelehnten Fächern.

15.1 Einleitung

Der Grader PABS (ProgrammierAufgabenBewertungsSystem) ist eine Eigenentwicklung des Instituts für Informatik der Universität Würzburg. Die Entwicklung begann im Jahr 2010, um das damals im Einsatz befindliche Praktomat zu ersetzen, dessen damalige Version einen umständlichen Upload der Lösungsdateien erforderlich machte und bei hoher Belastung Performance-Probleme generierte, so dass das Bestehen von Tests die in der Ausführungszeit limitiert waren (z. B. Aufgaben bestimmte Algorithmen effizient zu implementieren) zur Glückssache wurde.

Aus diesen Erfahrungen und dem Einsatzweck für das Java-Programmierpraktikum ergaben sich als Kriterien: Skalierbarkeit, eine (der Zielgruppe entsprechend) elegante Möglichkeit für den Upload der Lösungen und die Unterstützung von Sprachen, die auf der Java Virtual Machine [MD97] (JVM) basieren.

Dieses Kapitel basiert in weiten Teilen auf [Iff+15].

15.2 Technologie und Architektur

PABS besteht aus mehreren, miteinander interagierenden Modulen: Einer Webanwendung, dem zugehörigen SVN Repository und mehreren Aktoren, welche in einem Master-Worker-Konzept angeordnet sind.

Die Webanwendung dient dazu, mit den Studierenden zu interagieren. Sie gewährt Zugang zu der Aufgabenstellung (siehe Abbildung 15.1) und zeigt den Studierenden eine Liste der hochgeladenen Lösungen mit der jeweiligen Einstufung der Abgabe (nicht bewertet, abgelehnt, akzeptiert, als zu bewertende Lösung markiert) an. Ist noch keine Bewertung erfolgt (automatisches Bewerten beim Hochladen kann vom Kursadministrator selektiv pro Aufgabe aktiviert werden, siehe Abbildung 15.2), können die Studierenden über die Oberfläche diese auslösen. Bei Aufgaben, deren Bewertung durchgeführt wurde, können die Studierenden das generierte Feedback einsehen, wie in Abbildung 15.3 gezeigt. Abgaben, die als akzeptiert markiert sind, können als zu bewertende Lösung markiert werden. So bleibt es möglich, nach Einreichen einer gültigen Lösung noch Nachbesserungen (z. B. Beseitigung von Debug-Code oder Ergänzung weiterer Kommentare) durchzuführen. Abhängig von der Entscheidung des Kursadministrators kann, wenn keine Lösung markiert wurde, zum Zeitpunkt des Abgabetermins die zuletzt hochgeladene akzeptierte Abgabe gewählt werden. Diese Möglichkeit wurde integriert, da sich gezeigt hat, dass viele Studierende vergessen eine endgültige Lösung zu markieren.

Kursadministratoren können neue Aufgaben erstellen (siehe Abbildung 15.2), für diese Aufgaben sowohl einen Bereitstellungs- als auch einen Abgabezeitpunkt festlegen und entscheiden, ob bestimmte Bewertungen optional oder geheim (Ergebnisse nur für Lehrende sichtbar) sind. Kursadministratoren und Assistenten haben die Möglichkeit, die Bewertungen der Kursteilnehmer zu betrachten, z. B.

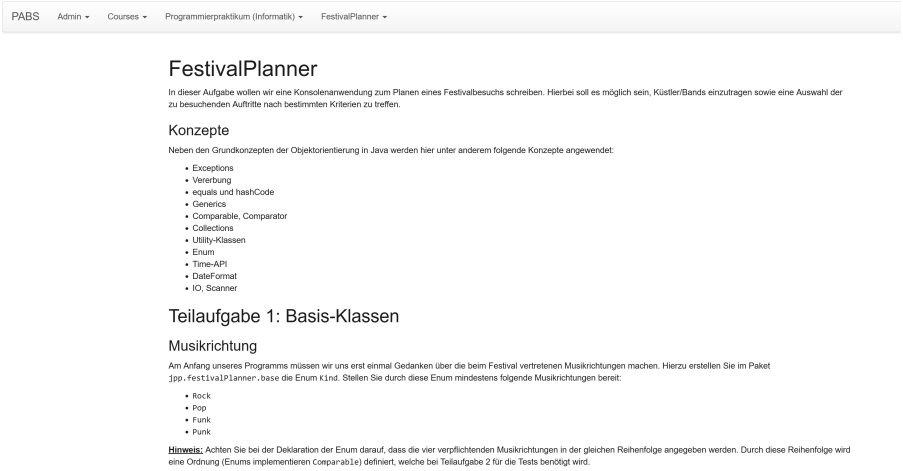

Abbildung 15.1: Die als XWIKI-Code bereitgestellte Aufgabenstellung wird den Studierenden im Webinterface angezeigt.

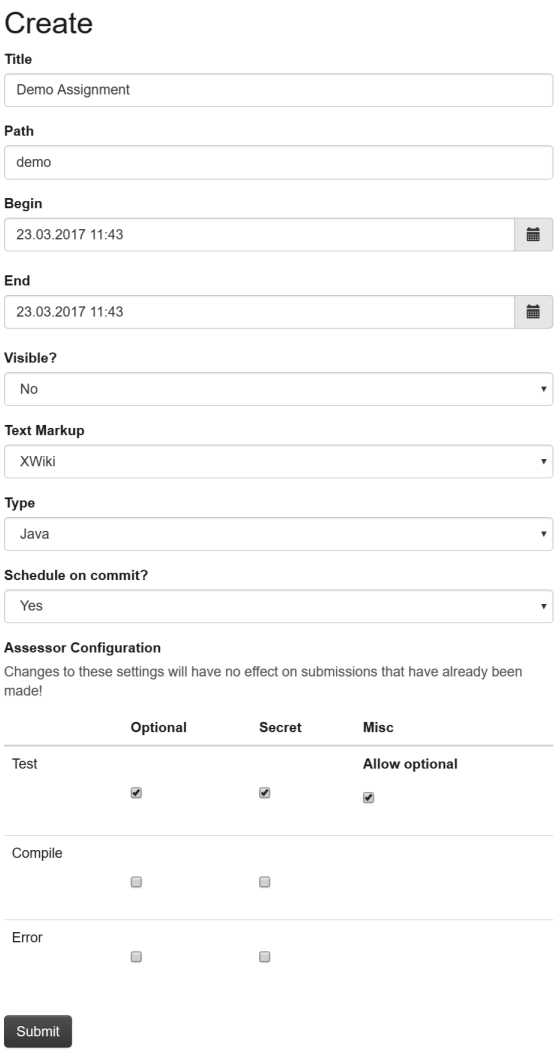

Abbildung 15.2: Kursadministratoren können beim Erstellen einer Aufgabe deren Titel [Title], Pfad im SVN [Path], Start [Begin] und Ende [end] angeben. Die Aufgabe kann zu Beginn auf unsichtbar [Visible?] gestellt werden. Es kann die Programmier- [Type] und Markupsprache [Text Markup] ausgewählt werden. Ob Aufgaben beim Commit bereits geprüft werden lässt sich konfigurieren [Schedule on commit?]. Unter Assessor Configuration lässt sich für Tests, Kompilierschritte und Fehlermeldungen wählen, ob es optionale und/oder versteckte Kriterien gibt.

zur manuellen Nachkorrektur oder um bei Problembeschreibungen in den Foren oder per E-Mail die Umstände besser nachvollziehen zu können. Auch ist es möglich, sich ausgeben zu lassen, welche Studierenden bereits eine Lösung markiert haben, wie in Abbildung 15.4 gezeigt.

Zur Einreichung der zu bewertenden Abgaben, aber auch zur Verwaltung der Aufgabenstellung, der Projektkonfiguration und der Bewertungsfälle wird Subversion[1] (SVN) genutzt. Für diese Zwecke wird pro Kurs ein SVN Repository erstellt. Im Repository werden die Konfiguration und die Abgaben in unterschiedliche Pfade getrennt. Studierende erhalten lediglich Zugriff auf ihren eigenen Ordner mit ihren Abgaben, während Tests, Konfiguration, etc. in anderen Ordnern liegen. Dadurch wird verhindert, dass Studierende den Quellcode von Blackboxtests einsehen oder gar manipulieren können.

Das SVN Repository ermöglicht Studierenden und Leherenden gleichermaßen einfachen Zugang zur Arbeitshistorie. Das ermöglicht Studierenden falsch eingeschlagene Lösungswege durch das Zurückspringen auf alte Versionen zu korrigieren. Lehrende auf der anderen Seite können so einfacher Feedback geben und werden bei neu aufgetretenen Problemen durch in SVN integrierte Tools (z. B. diff) unterstützt. Gleichzeitig ist es einfacher möglich Plagiate zu erkennen, da vor allem die Unterscheidung zwischen Plagiiertem und Plagiator erleichtert wird (letzterer hat in der Regel deutlich weniger Commits). Gegenüber der Speicherung der vollständigen Abgaben (z. B. in Datenbanken) hat die Speicherung in

Abbildung 15.3: Studierende können bestandene und nicht bestandene Tests einsehen. Zu letzteren wird das Feedback der TestSuite (hier JUnit) angezeigt.

1 https://subversion.apache.org

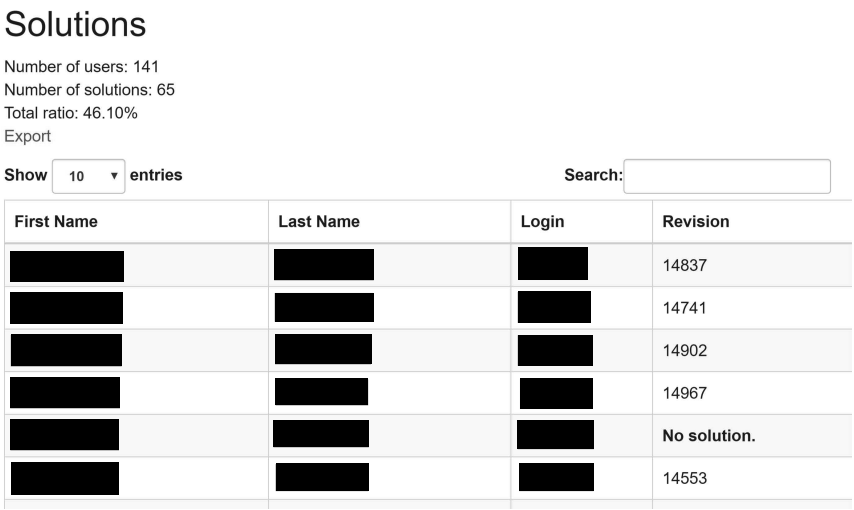

Abbildung 15.4: Betreuer können einsehen, welche und wie viele Studierende bereits Lösungen zur Abgabe markiert haben

einem SVN Repository technische Vorteile. Ein SVN Repository ist äußerst kompakt. Selbst Kurse mit vielen Studierenden und tausenden Commits sind selten größer als 100 MB. Weiterhin kann das Repository unabhängig von der Webanwendung genutzt werden, was ein Weiterarbeiten bei Ausfällen dieser ermöglicht. Auch ist den Studierenden und Betreuern so die Wahl der IDE oder gar der Verzicht auf eine solche freigestellt. Eine Diskussion, ob nun Eclipse oder IntelliJ eingesetzt werden soll, wird so elegant umgangen. Gleichzeitig verbleibt den Kursadministratoren trotzdem die Möglichkeit, als Gerüst einen vorbereiten Workspace einer Entwicklungsumgebung anzubieten. SVN fordert zwar eine gewisse Einarbeitung, unter Betrachtung des Einsatzszenarios für Studierende der Informatik und verwandte Fachrichtungen ist der Erwartungshorizont angemessen, dass Studierende sich selbstständig in Tools wie SVN hineinarbeiten können. Die Lernkurve wird dadurch etwas vereinfacht, dass es einen Schalter gibt, um bei Konflikten einfach das Repository zurück zu setzen, so dass Studierende, die nur an einem Gerät arbeiten, theoretisch alleine mit der Nutzung des Commit-Befehls auskommen können.

Die Generierung des Feedbacks kann abhängig von der Anzahl der Tests und der Codequalität eine zeit- und ressourcenaufwändige Aufgabe sein. Insbesondere komplexere Tests (z. B. unter Nutzung von Java-Reflection) können mehrere Sekunden beanspruchen. Um den Prozess der Feedbackgenerierung bei einer wachsenden Nutzerzahl zu unterstützen, nutzt PABS sog. Agenten, die mit der Web-

anwendung über das Netzwerk kommunizieren. Agenten können auf separaten Hosts ausgeführt werden und unabhängig voneinander aktiviert und deaktiviert werden. Während des Betriebs befindet sich ein Agent permanent in Kommunikation mit einer Warteschlange, die von der Webanwendung mit den zu bewerteten Abgaben befüllt wird. Dies ist in Abbildung 15.5 dargestellt. Eine in die Warteschlange eingestellte Abgabe wird an den ersten freien Agent weitergegeben, der diese in der Folge auswertet. Sobald das Feedback generiert wurde, wird dies an die Webanwendung zurück gesendet und der Warteschlange wird gemeldet, dass der Agent seine Arbeit abgeschlossen hat und wieder verfügbar ist.

Die Kommunikation zwischen Webanwendung und Agenten ist mittels Akka[2] realisiert. Zuerst wird ein Actor gestartet, der die Warteschlange verwaltet. Anschließend wird ein weiterer Actor gestartet, der Abgaben an die Warteschlange senden kann. Weiterhin wird eine beliebige Anzahl von Agent-Actors gestartert und beim Warteschlangen-Actor registriert. Sobald diese Pipline initialisiert wurde, kann damit begonnnen werden, Abgaben zu verarbeiten. Es ist sinnvoll anzumerken, dass das Feedback direkt an den Actor, der die Aufgabe in die Warteschlange eingestellt hat, weitergegeben und die Warteschlange so umgangen wird.

Der Agent nutzt Gradle[3] [MB11], um die Lösung der Studierenden zu kompilieren, sowie Tests und andere Analysetools auszuführen. Währenddessen sammelt ein spezialisiertes Plugin Informationen über den Build und stellt die Informationen dem Agent zur Verfügung. Der Agent beobachtet so den Build-Prozess und terminiert diesen wenn nötig (z. B. Endlosschleifen im abgegebenen Code). Nach einer erfolgreichen Ausführung des Builds erstellt der Agent die Feedbackdaten aus den Build-Informationen und sendet diese an die Webanwendung zurück.

15.3 Aufgabenstruktur

PABS baut auf dem Konzept eines Kurses auf. Ein Kurs besteht aus einer beliebigen Anzahl von Aufgaben und hat eine Anzahl an Studierenden, die in den Kurs eingeschrieben sind. Jede Aufgabe hat ein Zeitfenster, bis wann sie gelöst sein muss, um die eingereichte Lösung als gültig markieren zu können. Nachdem diese Zeit abgelaufen ist, kann die Aufgabe zwar weiter bearbeitet werden und Abgaben erhalten weiterhin Feeback, aber es ist nicht mehr möglich, eine endgültige Lösung abzugeben. Für jede Aufgabe kann vom Lehrenden gewählt werden, was nötig ist, damit die Abgabe vom System akzeptiert wird. So kann das Beste-

2 http://www.akka.io
3 http://www.gradle.org

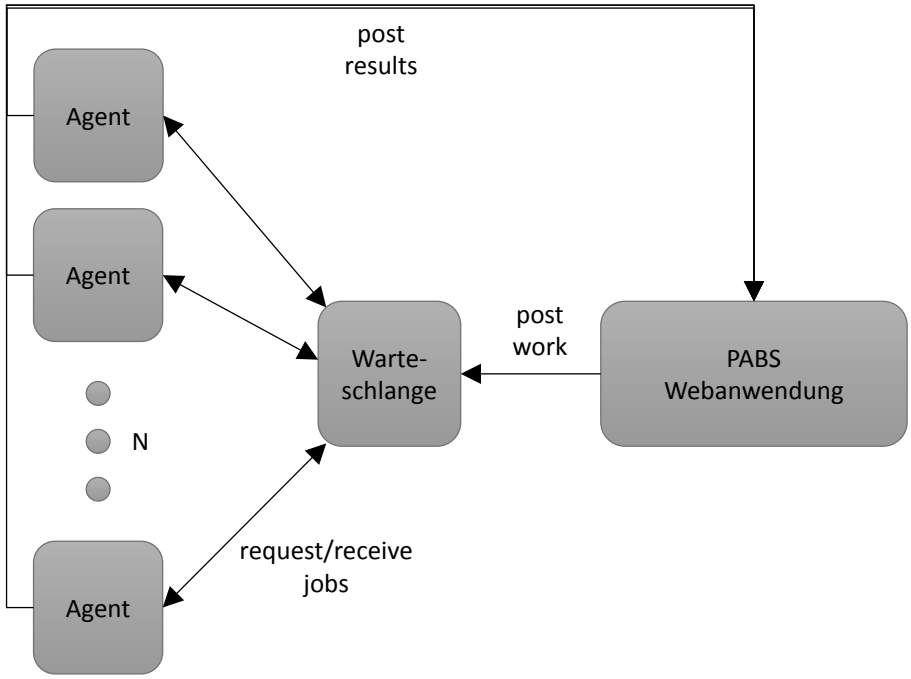

Abbildung 15.5: Zeigt die asynchrone Kommunikation zwischen der PABS Webanwendung und ihren Agenten durch eine Warteschlange. Zuerst fügt die Webanwendung eine Abgabe in die Warteschlange ein. Ein freier Worker-Agent fragt anschließend nach Arbeit. Ist der Agent mit seiner Arbeit fertig, sendet er die Ergebnisse an die Webanwendung zurück und informiert das Warteschlangensystem, dass die Abgabe verarbeitet wurde.

hen aller Tests verlangt werden, aber auch können bestimmte Tests als optional markiert werden oder sogar versteckt werden, so dass zwar die Lehrenden die Ergebnisse sehen, die Studierenden aber gezwungen werden selbst zu testen. Ggf. kann PABS auch so konfiguriert werden, dass es lediglich prüft, ob der eingereichte Quellcode überhaupt kompiliert und so als Filter eingesetzt werden kann, um Aufgaben, die syntaktisch falsch sind, auszusortieren.

Im Folgenden werden Teile des Aufbaus des SVN Repositories beschrieben. Das Verzeichnis ist zur Veranschaulichung in Abbildung 15.6 als Baumstruktur dargestellt.

Die Aufgaben befinden sich in je einem Unterverzeichnis des Verzeichnisses „assignments“, welches auf oberster Ebene des SVN Repositories zu finden ist.

Abbildung 15.6: Verzeichnisstruktur des PABS SVN Repositories

Im Ordner jeder Aufgabe finden sich die Ordner „text“, „template“ und „project“. Im Ordner „text“ findet sich eine XWiki-Datei[4], welche die Aufgabenstellung enthält. Im Ordner „template“ kann ein Musterverzeichnis angelegt werden, das in den Aufgabenverzeichnissen der Studierenden, die sich einschreiben, automatisch erstellt wird. So ist es möglich, Bibliotheken mit auszuliefern, die für ein lokales Testen benötigt werden. Einzelne Anfängerkurse liefern sogar vollständige Workspaces inklusive Konfiguration aus, um so die Einstiegsschwelle in die Programmierung zu reduzieren. Die für das Feedback relevanten Daten finden sich im Ordner „project“. Dort findet sich zuerst eine Konfigurationsdatei für Gradle, in der man die zu kompilierenden Pfade setzen kann. Hier lassen sich auch weitere Bibliotheken oder Plugins ergänzen. Die Tests finden sich anschließend in einem, in der Konfigurationsdatei spezifizierten, Unterordner. Die Tests lassen sich einfach in die Gruppen `required`, `optional` und `hidden` kategorisieren. Finden sich im Pfadnamen die Zeichenfolgen „optional“ oder „hidden“, werden die Tests entsprechend klassifiziert. Alle anderen Tests sind immer `required`. Ausnahmen bestehen, wenn versteckte und/oder optionale Tests in der Weboberfläche für die Aufgabe deaktiviert wurden, dann werden auch diese als `required` gesetzt. Diese einfache Struktur ermöglicht es, zusammen mit der Konfigurationsdatei auch Anforderungen umzusetzen, die über das einfache Abarbeiten von Unit-Tests hinausgehen (z. B. Stilprüfung, Ressourcenverbrauchsmessungen).

15.4 Grading-Methoden und Feedbackmöglichkeiten

Gegenwärtig verfügt PABS lediglich über die Möglichkeit, Bewertungsfälle als „bestanden“ oder „nicht bestanden“ zu markieren. Eine Bepunktung muss gegenwärtig noch manuell durchgeführt werden. Zur Bewertung können beliebige Verfahren herangezogen werden, solange sie auf der JVM laufen und in Gradle konfiguriert werden können.

Das Feedback findet in textueller Form statt. Dazu werden den Studierenden die bestandenen und nicht bestandenen Bewertungsfälle aufgelistet, sowie bei letzteren noch die Auswertung, warum der Fall nicht bestanden wurde. Bestandene und nicht bestandene Bewertungsfälle sind farblich entsprechend hervorgehoben.

Eine Plagiatserkennung ist derzeit als externes Modul an PABS angebunden, das manuell ausgelöst werden muss. Das Tool ist mit der Dateistruktur der PABS-Abgaben vertraut und kann so eine einfache Zuordnung der Plagiate ermöglichen.

4 http://www.xwiki.org

Nach dem Abschluss der Plagiatsprüfung wird eine Weboberfläche geboten, in der Paarungen ähnlicher Klassen aufgelistet werden. Die Similaritäten werden wie in Abbildung 15.7 graphisch aufbereitet dargestellt, so dass einfach zu erkennen ist, welche Teile übernommen wurden. Für die Erkennung werden unterschiedliche Methoden angewandt. Unter anderem werden durch Tokenizing [BTZ07] und die Anwendung von Metriken [Wha90] die beliebten Methoden der Umbenennung von Variablen oder der Umstellung der Methodenreihenfolge erkannt. Gegenwärtig wird das Modul um eine Erkennung von Plagiaten in den in Java 8 neu eingeführten Lambdas erweitert. Das Tool kann auch unabhängig von PABS genutzt werden, verliert dann aber einige Komfortfunktionen.

Bisher findet die Feedbackerstellung und die Bewertung nur im PABS-System direkt statt. Eine Integration für E-Learning-Plattformen wird angestrebt.

15.5 Bisheriger Einsatz

PABS wurde ursprünglich primär für die Unterstützung des Java-Programmierpraktikums für Studierende der Informatik (Bachelor Informatik, Bachelor Luft- und Raumfahrtinformatik und Lehramt) an der Universität Würzburg eingeführt. Die aktuelle Major Revision PABS 3 wurde zu diesem Zweck inzwischen sechsmal erfolgreich angewendet. Während zu Beginn etwa 60 Kursteilnehmer zu betreuen waren, ist die Zahl der Teilnehmer pro Semester inzwischen auf etwa 150 gewachsen.

Nach ersten erfolgreichen Tests wurde das Programm auch für das etwas einfacher ausgerichtete Praktikum für die Bachelorstudiengänge Wirtschaftsinformatik, Wirtschaftsmathematik, Computational Mathematics und Mensch-Computer-Systeme eingeführt. Hier finden sich pro Semester inzwischen etwa 200 Studierende.

Trotz seiner ursprünglichen Orientierung an den Praktika wird PABS in immer mehr Vorlesungen unterstützend zum Übungsbetrieb eingesetzt. Besonders sind hier die Vorlesungen Algorithmen und Datenstrukturen (ca. 400 Teilnehmer), Grundlagen der Programmierung (ca. 170 Teilnehmer) und Informatik für Hörer anderer Fakultäten (ca. 90 Teilnehmer) zu nennen. Zuletzt wurde PABS auch in dem Vorkurs für neue Studierende (ca. 130 Teilnehmer) eingesetzt. In diesem wird inzwischen auch der Umgang mit dem SVN erläutert. Für weitere Veranstaltungen wird der Einsatz geplant oder evaluiert.

Betrachtet man die Kennzahl Studierende mal belegter Kurse, so wird derzeit im Sommersemester ein Wert von etwa 570 und im Wintersemester von etwa 1.140 erreicht. Die Schwankung ist dadurch erklärbar, dass die oben genannten

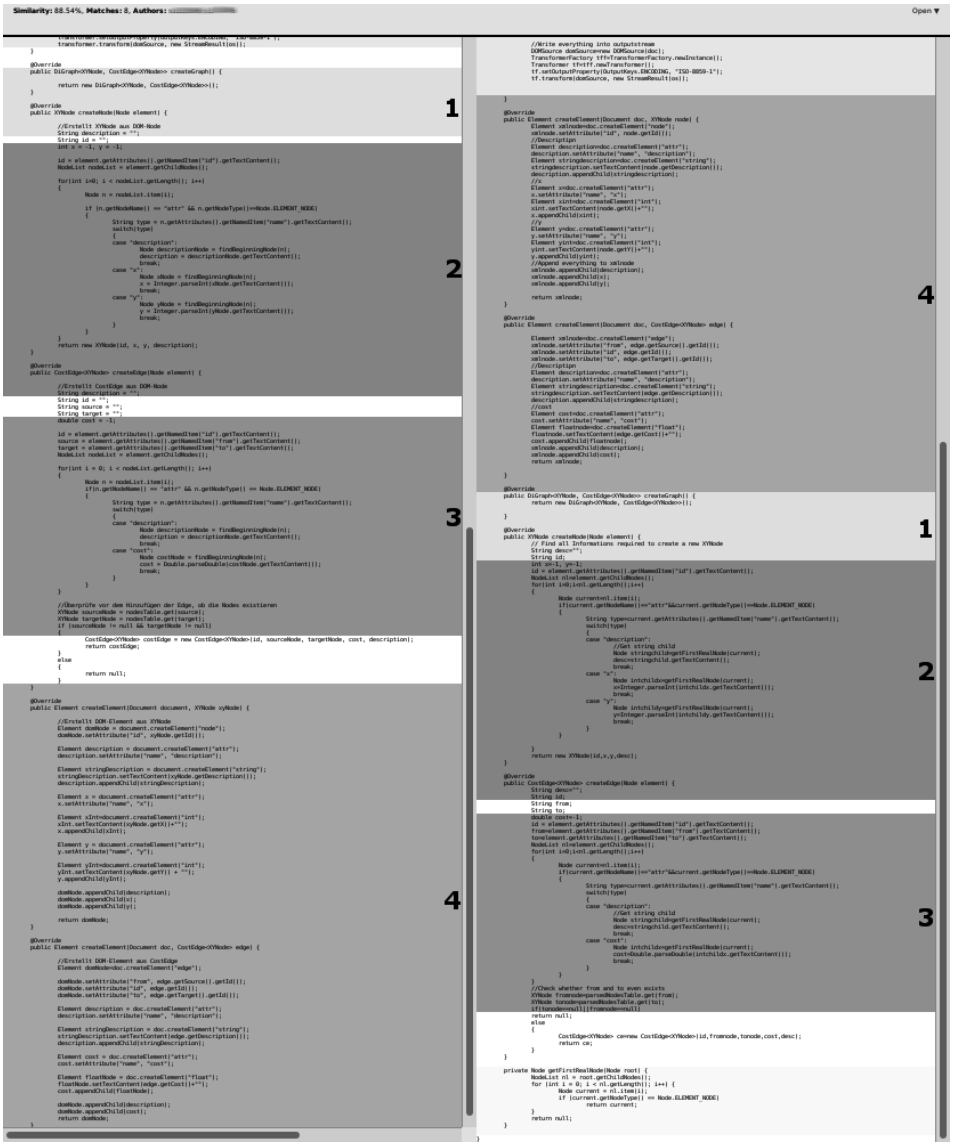

Abbildung 15.7: Bei der Plagiatsprüfung werden als ähnlich erkannte Bereiche in einer Splitansicht mit gleichen Farben/Zahlen hervorgehoben, so dass Plagiate einfach und schnell erkannt werden können.
Im Beispiel hat das Programm zwei Abgaben zu 88% als übereinstimmend erkannt. Es ist zu erkennen, dass zur Verschleierung des Plagiates die Reihenfolge der Blöcke geändert wurde (der letzte Block auf der linken Seite wurde nach vorne gezogen), sowie teilweise Leerzeilen entfernt wurden.

großen Vorlesungen nur im Winter angeboten werden. Derzeit sind in PABS 1.738 Nutzer eingeschrieben, die zum großen Teil mehrere Kurse belegen oder belegt haben.

15.6 Verfügbarkeit und Ausblick

> Wer die geschützten Testfälle knacken kann, dem gebührt es auch, das Praktikum zu bestehen.
>
> – Prof. Dr. Jürgen Wolff von Gudenberg

PABS ist derzeit noch als Closed Source in Entwicklung, da noch einige Sicherheitsaspekte geklärt bzw. verbessert werden müssen. Unter den wissenschaftlichen Hilfskräften, die die Praktika betreuen, hat sich ein kleiner Wettbewerb entwickelt, um durch Exploits mit Studentenrechten an die Quellen der Testfälle zu kommen. Zwar ist dem obigen Zitat des Dozenten, der lange Jahre das Programmierpraktikum organisiert und verantwortet hat, zuzustimmen. Dies gilt allerdings nur für den ersten Studierenden, der diese Schranke umgeht, während dann die weiteren die automatischen Tests durch Code umgehen können, der exakt die Lösungen der getesteten Fälle zurückliefert. Sobald die bekannten Sicherheitslücken beseitigt sind, ist geplant, das Programm unter einer OpenSource-Lizenz über eine öffentlich zugängliche Platform zur Verfügung zu stellen. An der Entwicklung interessierte Personen können aber bereits jetzt einen Zugang zu den Repositories erhalten.

In Zukunft sind viele Weiterentwicklungen für PABS geplant, die sich großteils in den Bereichen der Unterstützung weiterer Funktionen, die Vereinfachung der Nutzung und der Erhöhung der Sicherheit ansiedeln.

Während bisher primär Java implementiert wurde, soll die Unterstützung von anderen Sprachen auf Basis der JVM erweitert werden. Während für Scala und Groovy bereits Musterbeispiele vorliegen, soll demnächst der JVM Python-Interpreter Jython[5] integriert werden, da geplant ist, die Sprache Python aufgrund ihrer einfachen Syntax bei Vorlesungen für Hörer außerhalb der Informatik einzusetzen. Auch die Unterstützung von C und C++ ist angedacht, um in Vorlesungen, die hardwarenahes Programmieren lehren, eine Feedbackmöglichkeit zu schaffen. Besonders bei diesen beiden Sprachen müssen allerdings zusätzliche Sicherheitsmaßnahmen ergriffen werden, da sie aufgrund ihrer Hardwarenähe zusätzliche Angriffsvektoren bieten.

5 http://www.python.org

Aus den genannten Sicherheitsgründen, aber auch zur Verbesserung der Elastizität [HKR13] sollen die die Bewertung ausführenden Agenten auf eine Containerarchitektur umgestellt werden. Dadurch wird eine Kapselung erreicht, so dass Rechte-Exploits des ausgeführten Codes keinen Schaden außerhalb anrichten können. Weiterhin soll so innerhalb der Cloud Environments der Universität ermöglicht werden, auf Lastspitzen (z. B. während der Praktikumstutorien) flexibel durch das Ausbringen zusätzlicher Agenten zu reagieren.

Die Verwendung von PABS soll vor allem für die Lehrenden vereinfacht werden. Dazu soll unter anderem für die Bearbeitung der Aufgabenstellungen zusätzlich zum Upload des XWIKI-Quellcodes über das SVN direkt über eine Weboberfläche ermöglicht werden, die einen WYSIWYG-Editor (What You See Is What You Get) bereitstellt. Auch sollen Lehrende nicht mehr manuell die Plagiaterkennung anstoßen müssen, sondern nach Abgabenende automatisch über solche informiert werden. Zur Reduzierung von Redundanzen soll der Bewertungsexport in Moodle realisiert werden.

Auch das Bewertungskonzept in bestanden und nicht bestanden soll um die Möglichkeit der Punktevergabe erweitert werden. Dazu ist eine Annotierung der Testfälle angestrebt, so dass jedem Testfall eine Punktezahl zugeordnet werden kann.

15.7 Abschluss

Auf den vergangenen Seiten wurde der Grader PABS vorgestellt. Der Grader ist auf Skalierbarkeit ausgelegt und hat eine stetig wachsende Nutzerbasis. Die Funktionen befinden sich weiter in der Entwicklung und werden regelmäßig erweitert.

Literatur für dieses Kapitel

[BTZ07] Steven Burrows, Seyed M.M. Tahaghoghi und Justin Zobel. „Efficient plagiarism detection for large code repositories“. In: *Software-Practice and Experience* 37.2 (2007), S. 151–176.

[HKR13] Nikolas Roman Herbst, Samuel Kounev und Ralf Reussner. „Elasticity in cloud computing: What it is, and what it is not“. In: *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. 2013, S. 23–27.

[Iff+15] Lukas Iffländer u. a. „PABS – a Programming Assignment Feedback System“. In: *Workshop „Automatische Bewertung von Programmierauf-*

gaben“ (ABP 2015). Bd. 1496. CEUR Workshop Proceedings. Nov. 2015.

[MB11] Matthew McCullough und Tim Berglund. *Building and Testing with Gradle*. O’Reilly Media, Inc., 2011.

[MD97] Jon Meyer und Troy Downing. *Java virtual machine*. O’Reilly & Associates, Inc., 1997.

[Wha90] Geoff Whale. „Software metrics and plagiarism detection“. In: *Journal of Systems and Software* 13.2 (1990), S. 131–138.