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

13 Der Grader GATE

Oliver Müller und Sven Strickroth

Zusammenfassung
Das an der TU Clausthal entstandene System GATE *(Generic Assessment & Test Environment) wurde insbesondere zur Verbesserung der Ausbildung im Bereich der Java-Programmierung und zur Unterstützung von Tutorinnen und Tutoren in Java-Programmierübungen mit mehreren hundert Studierenden entwickelt.* GATE *ist webbasiert, plattformunabhängig und in Java geschrieben. Dieses Kapitel geht zum einen auf die Funktionalität des* GATE*-Systems ein und stellt zum anderen dessen Architektur näher vor. Zum Abschluss folgt eine Zusammenfassung wesentlicher Evaluationsergebnisse zum* GATE*-System sowie eine Übersicht über dessen Nutzerzahlen.*

13.1 Einleitung

GATE ist ein Online-Abgabesystem mit integrierter LMS (Learning Management System)-Funktionalität, das verschiedene Funktionen zur Organisation und Verwaltung von Praktika oder vorlesungsbegleitenden Übungen bereitstellt.

Das System wurde v. a. für den Einsatz im Übungsbetrieb von Einführungsveranstaltungen zur Java-Programmierung an Universitäten entwickelt, an denen häufig eine größere Anzahl an Studierenden teilnimmt. Konkret wird das System zurzeit regelmäßig im Rahmen des Übungsbetriebs zweier Veranstaltungen dieser Art verwendet. Bei diesen handelt es sich um einen einführenden Java-Programmierkurs für Studierende der Betriebswirtschaftslehre (ca. 200–300 Studierende) und um einen Java-Programmierkurs für Studierende der (Wirtschafts-) Informatik im zweiten Semester (ca. 80–100 Studierende).

Teile dieses Kapitels entstanden im Rahmen des Projekts eCult, Teilvorhaben eAssessment, gefördert durch das Bundesministerium für Bildung und Forschung unter dem Förderkennzeichen 01PL16066L. Die Verantwortung für den Inhalt dieses Kapitels liegt bei den Autoren.

Um den tendenziell recht hohen Bewertungs- und Korrekturaufwand für Tutorinnen und Tutoren in für diese Veranstaltungen durchgeführten Programmierübungen zu reduzieren, bietet das System Funktionen an, die Tutorinnen und Tutoren insbesondere bei der Korrektur und Bewertung von Aufgaben aus den Bereichen der Java-Programmierung und Modellierung von UML-Klassen- und Aktivitätsdiagrammen unterstützen. Zu Zwecken des Self-Assessment können einige dieser Funktionen auch Studierenden zur Verfügung gestellt werden, mit dem Ziel, die Qualität studentischer Abgaben zu erhöhen.

13.2 Funktionen des GATE-Systems

In den folgenden Abschnitten werden die wichtigsten Funktionen des GATE-Systems erläutert. Abschnitt 13.2.1 geht insbesondere auf den in Abbildung 13.1 gezeigten Anwendungsfall *Aufgaben verwalten* ein. In Abschnitt 13.2.2 wird auf die Möglichkeit eingegangen, in GATE Aufgaben mit randomisierten Werten (dynamische Aufgaben) anzulegen. Abschnitt 13.2.3 beschäftigt sich mit den im GATE-System zur Verfügung stehenden Funktionstests (*Funktionstests definieren*) sowie mit der Frage, wie in GATE Lösungen zu Aufgaben bewertet werden (u. a. *Punkte vergeben*, *Abgaben überprüfen*). Abschnitt 13.2.4 befasst sich mit den Möglichkeiten, Studierenden sowohl textuelles Feedback (*Textuelles Feedback geben*) als auch automatisiert generiertes Feedback über GATE zur Verfügung zu stellen, während Abschnitt 13.2.5 auf die im System zur Verfügung stehende Funktion zur Plagiatserkennung (*Plagiatserkennung konfigurieren*) eingeht.

13.2.1 Funktionen zur Übungsorganisation und -verwaltung

Die wesentlichen Funktionen zur Organisation und Verwaltung von Übungen können wie in Abbildung 13.1 gezeigt ausschließlich von Betreuern einer Veranstaltung genutzt werden. Diese können also u. a. für Teilnehmer einer Veranstaltung Tutorenrechte vergeben (*Tutorenrechte vergeben*) sowie Übungsgruppen anlegen und bearbeiten (*Gruppen verwalten*). Eine weitere zentrale Funktion stellt in diesem Zusammenhang das Anlegen und Bearbeiten von Aufgaben dar (*Aufgaben verwalten*), für die Studierende Lösungen im GATE-System einreichen sollen (*Lösungen einreichen*).

Beim Anlegen einer Aufgabe in GATE lassen sich in einem *ersten Schritt* die im Folgenden aufgeführten Aspekte konfigurieren bzw. angeben:

Allgemeine Informationen zur Aufgabe Für jede Aufgabe lässt sich ein Aufgabentitel und die Aufgabenstellung angeben bzw. formulieren. Jede Aufgabe kann einer von einem Betreuer zuvor angelegten Aufgabengruppe zugeordnet werden, die üblicherweise einem Aufgabenzettel entspricht. Soll eine Aufgabe mit randomisierten Werten angelegt werden, so kann für diese ein in GATE zur Verfügung stehender Aufgabentyp ausgewählt werden.

Form der Abgabe und Bearbeitungszeitraum Das Einreichen einer Lösung zu einer Aufgabe kann in GATE entweder durch Eingabe dieser in ein durch das System bereitgestelltes Textfeld oder durch einen Dateiupload erfolgen. Die erstgenannte Möglichkeit ist für Theorie/Freitext- sowie dynamische Aufgaben vorgesehen. In das bereitgestellte Textfeld können z. B. nicht nur reine Ergebnisse, sondern auch ausführlichere Erläuterung/Begründungen zu Ergebnissen eingetragen werden. Soll zur Abgabe einer Lösung ein Dateiupload vorgenommen werden, so ist es erforderlich, einen regulären Ausdruck anzugeben, der die möglichen Dateinamen für die hochzuladenden Dateien festlegt. Diese Vorgabe ist z. B. für das Einreichen von Java Dateien sinnvoll, um zu verhindern, dass Studierende Java-Programme lediglich als Bytecode einreichen oder diese auf Grund

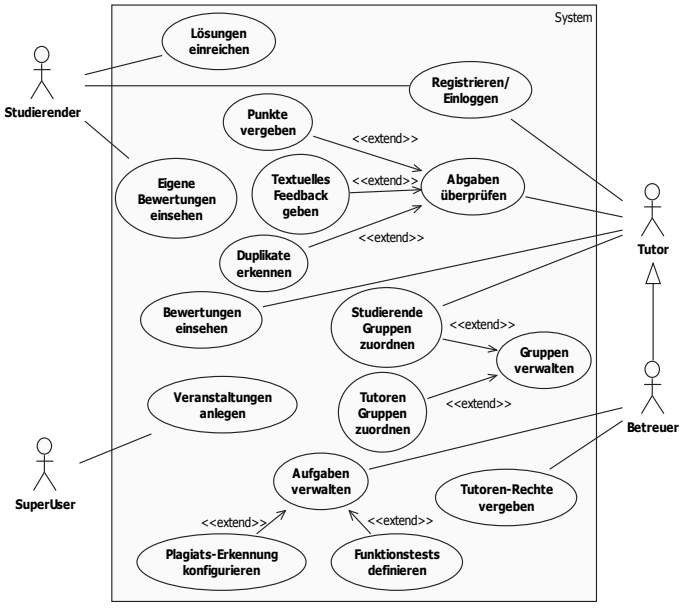

Abbildung 13.1: GATE: Funktionen des Systems und zur Ausführung berechtigte Personengruppen nach [Str09]

eines falsch gewählten Dateinamens nicht ausführbar sind. Ebenso ist es möglich Archive, bestehend aus mehreren Dateien, hochzuladen, die bei entsprechender Konfiguration automatisch entpackt werden. Zudem lässt sich angeben, welche Dateien für einen Tutor automatisch in der Ansicht für Bewertung und Korrektur einer Abgabe hervorgehoben werden sollen.

Das GATE-System lässt neben Einzelabgaben auch Abgaben in Gruppen zu. Die maximale Größe einer Abgabegruppe kann dabei vom Betreuer einer Veranstaltung für jede Aufgabe konfiguriert werden.

Weiterhin kann für eine Aufgabe definiert werden, in welchem Zeitraum eine Abgabe einer Lösung in GATE möglich ist. Innerhalb des festgelegten Abgabezeitraums lassen sich Abgaben stets erweitern bzw. eingereichte Dateien durch überarbeitete/korrigierte Versionen beliebig oft ersetzen. Dies ist u. a. notwendig, um Studierenden eine sinnvolle Nutzung der bereits erwähnten Self-Assessment Funktionalität zu ermöglichen.

Betreuer einer Veranstaltung können optional für jede Aufgabe erlauben, dass von Studierenden für die Lösung einer Aufgabe erstellte Dateien auch von Tutorinnen und Tutoren in das GATE-System hochgeladen werden können. Dies kann z. B. dann hilfreich sein, falls Studierende, die das System zum ersten Mal verwenden, auf Probleme beim Dateiupload stoßen oder aus Kulanz eine Abgabe in Einzelfällen auch nach der Abgabefrist erlaubt wird.

Spezifikation der Punktevergabe Für jede Aufgabe lässt sich die maximal zu erreichende Punktzahl definieren sowie festlegen, ob nur volle Punkte oder auch halbe Punkte, viertel Punkte etc. vergeben werden können. Zudem kann bestimmt werden, wann Studierende für ihre Abgaben zu einer Aufgabe die jeweils vergebenen Punkte im GATE-System angezeigt bekommen. Die Übersicht über die erreichten Punkte kann entweder manuell oder automatisch zu einem vorgegebenen Zeitpunkt für die Studierenden sichtbar geschaltet werden. Die Wahl der erstgenannten Option bietet sich an, wenn eine erfolgreiche persönliche Abnahme einer Lösung durch Tutorinnen/Tutoren erforderlich ist, damit den jeweiligen Studierenden die für ihre Lösungen bereits im System eingetragenen Punkte zugesprochen werden. Bei letztgenannter Option werden die Punkte für alle Studierenden gleichzeitig sichtbar.

In einem *zweiten Schritt* können für eine anzulegende Aufgabe beliebig viele Bewertungskategorien/-kriterien hinzugefügt werden (inkl. Kategorien für Bonuspunkte), für die sich jeweils die maximal zu erreichenden Punkte festlegen lassen.

Außerdem besteht die Möglichkeit, Musterlösungsdateien, die nur für Tutorinnen und Tutoren bzw. Betreuer der jeweiligen Veranstaltung einsehbar sind sowie Dateien, die Studierenden zur Bearbeitung einer Aufgabe bereitgestellt werden sollen, für eine Aufgabe im GATE-System hochzuladen.

Zu guter Letzt lassen sich Funktionstests (siehe Abschnitt 13.2.3) und Plagiatstests (siehe Abschnitt 13.2.5) konfigurieren. Die Tests sollen einerseits Tutorinnen und Tutoren bei der Bewertung von Lösungen unterstützen. Andererseits können Studierende innerhalb der Abgabefrist, sofern vorgesehen, Feedback vom GATE-System zu ihren eingereichten Lösungen über die zur Verfügung gestellten Funktionstest erhalten (für Details siehe Abschnitt 13.2.4).

13.2.2 Aufgaben mit randomisierten Werten

GATE unterstützt prototypisch das Anlegen von Aufgaben mit randomisierten Werten [MS13]. Die Funktion steht zurzeit nur für einige spezielle, vorgegebene Aufgabentypen zur Verfügung. Bei diesen handelt es sich im Wesentlichen um theoretische Aufgaben z. B. zur Berechnung von Speicherplatz bzw. Dateigrößen oder um Aufgaben zum Überführen einer Zahl in ein anderes Zahlensystem.

Bei Aufgaben mit randomisierten Werten wird Studierenden die gleiche Aufgabe mit jeweils individuellen Wertebelegungen für bestimmte Variablen präsentiert. Dies dient u. a. der Vermeidung möglicher Plagiate, da sich Studierende so über Lösungswege austauschen können, aber alle Studierenden erforderliche Berechnungen selbstständig durchführen müssen.

Die Abgabe einer Lösung zu einer Aufgabe mit randomisierten Werten erfolgt über separate Textboxen in GATE, in die der Rechenweg bzw. Zwischenlösungen sowie das von GATE automatisiert überprüfbare Endergebnis eingetragen werden können. Tutorinnen und Tutoren bekommen zu Korrekturzwecken nicht nur die Eingaben der Studierenden, sondern auch die jeweiligen individuellen Wertebelegungen und die für diese Werte vom System automatisch berechneten korrekten Zwischen- und Endergebnisse angezeigt. Somit können Tutorinnen und Tutoren Berechnungen der Studierenden auch im Fehlerfall nachvollziehen, ohne diese mit den individuellen Werten für alle Studierenden manuell durchführen zu müssen.

13.2.3 Grading-Methoden

Hinsichtlich der Korrektur und Bewertung von Lösungen zu in GATE erstellten Aufgaben verfolgt das System einen semiautomatischen Ansatz. Der manuelle Korrektur-/Bewertungsaufwand wird hierbei durch die Bereitstellung automatisch ausführbarer Tests, mit denen überprüft werden kann, ob eine Lösung bestimmte Kriterien (z. B. Kompilierbarkeit eines Java-Programms) erfüllt, verringert. Da die Tutorinnen und Tutoren die abschließende Bewertung bzw. Punktevergabe manu-

ell vornehmen müssen, ist trotzdem eine ganzheitliche Bewertung einer in das GATE-System eingereichten Lösung möglich (vgl. [AM05]).

Ergebnisse der in GATE zur Verfügung stehenden Tests können den für die Bewertung und Korrektur zuständigen Personen nach Ablauf der Frist für die Abgabe einer Lösung zu einer Aufgabe automatisch zur Verfügung gestellt werden. Damit können Tutorinnen und Tutoren schneller sehen, ob eine Lösung korrekt ist oder nicht und somit einfacher zielgerichtetes Feedback für die Lösung vergeben.

Das GATE-System bietet die im Folgenden erläuterten Testmethoden für Java-Programme bzw. UML-Aktivitäts- und Klassendiagramme an.

Compile-/Syntaxtest für Java-Programme Zur Überprüfung eines eingereichten Java-Programms auf syntaktische Korrektheit werden von GATE ein Kompiliervorgang für das Programm angestoßen und die Ausgaben aufgezeichnet.

UML-Vergleichstest Mit Hilfe dieser Art von Test können UML-Klassen- und Aktivitätsdiagramme überprüft werden, die von Studierenden mit einer in GATE integrierten, modifizierten und über Java Webstart ausführbaren Version des Tools Argo-UML[1] erstellt wurden [Sch+12]. Die Prüfmethode basiert dabei auf einem Vergleich der erstellten Diagramme mit einer Musterlösung, die beim Anlegen eines UML-Vergleichstests in GATE hochgeladen werden muss.

Tests auf funktionale Korrektheit von Java-Programmen Da bei dieser Art von Tests eingereichte Java-Programme innerhalb des GATE-Systems ausgeführt werden, kann für diese ein Timeout festgelegt werden, der dafür sorgt, dass die Ausführung nach einer vorgegebenen Zeit abgebrochen wird.

JUnit-Test: Die funktionale Korrektheit eines Java-Programms kann in GATE durch selbst definierte JUnit-Tests überprüft werden. Zum Anlegen eines JUnit-Tests in GATE wird eine .jar Datei, die den kompilierten Test enthält, in das GATE-System hochgeladen. In diesem Zusammenhang ist zudem die Angabe der Main-Testklasse erforderlich. Wird z. B. eine modifizierte Java-JOptionPane Klasse bereitgestellt, so unterstützt GATE auch das automatisierte Testen von Java-Programmen, die GUI (Graphical User Interface) Funktionen nutzen.

Regexp-Test: Bei dieser Art von Test wird ein eingereichtes Java-Programm, optional mit Kommandozeilenparametern, in GATE ausgeführt. Die Ausgabe des Programms wird dabei mit einem regulären Ausdruck verglichen,

1 http://argouml.tigris.org/

der bei der Konfiguration des Tests neben den Kommandozeilenparametern und dem vorzugebenden Namen der Main-Klasse des Programms angegeben wird.

13.2.4 Feedback für Studierende

Feedback für Studierende kann in GATE sowohl manuell durch Tutorinnen und Tutoren bzw. Betreuer vergeben, als auch über die in Abschnitt 13.2.3 aufgeführten Testmethoden automatisiert generiert und bereitgestellt werden. GATE bietet dabei verschiedene Möglichkeiten das Feedback einzuschränken. Zum einen kann die Anzahl der Feedbackanforderungen limitiert werden, um z. B. Gaming-the-System-Ansätze von Studierenden zu unterbinden (vgl. [Bak+05; Bak+08]), zum anderen ist es auch möglich, die Rückmeldung für die Studierenden auf eine einfache „bestanden/nicht bestanden“ Wertung statt detaillierter Ausgaben einzuschränken (vgl. [SOP11]).

Manuell vergebenes Feedback Für Feedback, das manuell vergeben werden soll, steht in GATE eine Freitextkommentarfunktion zur Verfügung. Das Feedback kann dabei an Studierende zur genaueren Erläuterung der Korrektur und Bewertung ihrer Abgaben gerichtet sein. Zudem existiert eine Möglichkeit der abgabenbezogenen internen Kommunikation zwischen den Tutorinnen und Tutoren und Betreuern einer Veranstaltung, die z. B. dazu genutzt werden kann, sich über vermutete Plagiate oder bezüglich Unklarheiten bei der Korrektur und Bewertung einer konkreten Abgabe auszutauschen.

Feedback für UML-Klassen- und Aktivitätsdiagramme Wie bereits in Abschnitt 13.2.3 erwähnt, kann direkt aus GATE eine erweiterte Version des Tools Argo-UML aufgerufen werden. Mit dieser lassen sich Lösungen zu einer in GATE angelegten UML-Modellierungsaufgabe erstellen und im XMI Format nach GATE exportieren. Möchte ein Studierender eine bereits exportierte Abgabe bearbeiten, so wird diese automatisch von der modifizierten Argo-UML-Version abgerufen und angezeigt. Das modifizierte Tool bietet zudem einen Feedback-Button (s. Abbildung 13.2), über den Studierende spezielles Feedback zu ihren nach GATE exportierten UML-Klassen- bzw. Aktivitätsdiagrammen anfordern können [Sch+12].

Das Feedback wird über den in GATE implementierten UML-Vergleichstest automatisiert generiert und nach Anforderung in Argo-UML in einem separaten Fenster in textueller Form ausgegeben (s. Abbildung 13.2). Konkret wird dabei

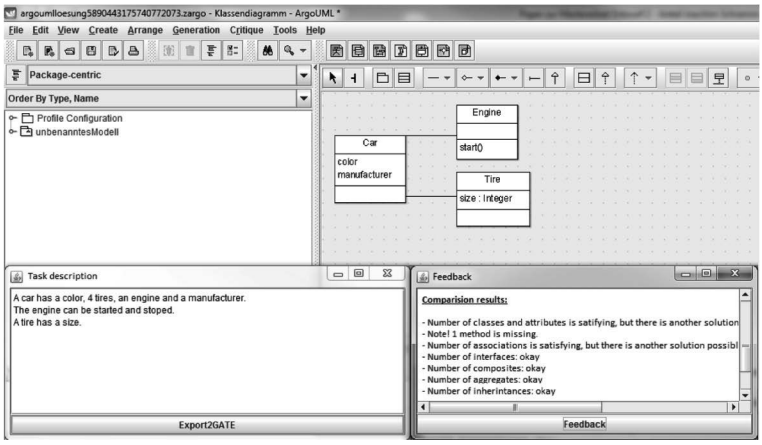

Abbildung 13.2: Feedback mit Argo-UML [Sch+12]

z. B. für jeden durch den UML-Vergleichstest berücksichtigten Elementtyp eines UML-Klassen- bzw. Aktivitätsdiagramms (vgl. [Sch+12], S. 3) angezeigt, ob im Vergleich zur Musterlösung zu wenige, zu viele oder eine richtige Anzahl von Elementen des jeweiligen Typs in der zu prüfenden Modellinstanz enthalten sind.

Automatisiert generiertes Feedback für Java-Programme Diese Art Feedback kann in GATE durch den in Abschnitt 13.2.3 erläuterten Compile-/Syntaxtest bzw. durch die im selben Abschnitt beschriebenen JUnit- und Regexp-Tests bereitgestellt werden. Als Feedback wird dabei angezeigt, ob ein Test erfolgreich war oder nicht. Zusätzlich können weitere Hinweise (wie z. B. Meldungen des Compilers oder der Tests) ausgegeben werden. Als Ergebnis eines nicht erfolgreichen Regexp-Tests wird lediglich die Ausgabe des getesteten Java-Programms sowie das Testergebnis angezeigt.

13.2.5 Plagiatserkennung

Als generelles Problem bei der Bearbeitung von Übungsaufgaben haben sich Plagiate gezeigt. Daher wurde eine Unterstützung für das übungsgruppenübergreifende Auffinden von Plagiaten in GATE integriert. Insgesamt stehen hierfür aus Gründen der Flexibilität drei verschiedene Algorithmen zur Verfügung (vgl. [Str09]), die sich jeweils für spezielle Einsatzzwecke eignen und auch gleichzeitig ausgeführt werden können. Bei der Ausführung eines Plagiatstests werden alle Abgaben zu einer Aufgabe nach einer Normalisierung paarweise miteinander vergli-

chen. Dabei kann, abhängig vom verwendeten Algorithmus, aus einer vorgegebenen Menge von möglichen Normalisierungen gewählt werden (z. B. Entfernen von Leerzeichen oder Kommentaren, Konvertierung zu Kleinbuchstaben, etc.). Als Ergebnis eines Plagiatstests wird für jede Abgabe in GATE für die Tutorinnen und Tutoren die Information hinzugefügt, zu welchen anderen Abgaben eine Ähnlichkeit besteht. Der Grad der Ähnlichkeit wird dabei in Prozent angegeben. Für einen Plagiatstest kann die minimale Ähnlichkeit konfiguriert werden, die jeweils zwei Abgaben zueinander aufweisen müssen, damit sie als hinreichend ähnlich gelten und somit im GATE-System aufgeführt werden. Ebenso können Dateien bestimmt werden, die durch den jeweiligen Plagiatstest nicht berücksichtigt werden sollen (z. B. vorgegebene Vorlagen oder automatisch generierte Dateien der verwendeten Entwicklungsumgebung).

Insbesondere für die Bewertung von möglichen Plagiaten gibt es in GATE eine Funktion, um eingesandte Quelltexte ohne Kommentare anzuzeigen. Tutorinnen und Tutoren können somit z. B. verhindern, dass Studierende, deren Abgaben unter Plagiatsverdacht stehen, bei der Präsentation ihrer Lösungen auf Kommentare zurückgreifen können, die nicht von ihnen selbst verfasst wurden.

Plaggie-Test Für diesen Test wurde die Standalone-Plagiatserkennungsengine *Plaggie* [ASR06], bei der es sich um eine frei verfügbare Variante von JPlag [PMP00] handelt, in GATE integriert. Über Plaggie lässt sich der Source-Code von Java-Programmen miteinander vergleichen, um automatisiert Plagiate in einer Menge von Java-Programmen entdecken zu können. Der Test ist besonders dafür geeignet, Ähnlichkeiten zwischen komplexeren Java-Programmen zu erkennen [SOP11; LC04].

Levenshtein-Test Neben der auf Java spezialisierten Plagiatserkennungsengine Plaggie wird eine Ähnlichkeitsbestimmung zweier textbasierter Abgaben mit Hilfe der Berechnung der *Levenshtein-Distanz* [Lev66] durchgeführt. In diesem Zusammenhang kann zudem in einem gewissen Rahmen festgelegt werden, welche Teile einer Lösung getestet werden sollen (nur Kommentare, nur Quellcode oder beides?). Da insbesondere durch die Berechnung der Levenshtein-Distanz keine Methodenpermutationen erkannt werden können, eignet sich der Einsatz des Levenshtein-Tests vor allem für kleinere Aufgaben (wie z. B. Aufzählen der Zahlen von 1 bis 10 in einer Schleife) bei denen Plaggie größere Übereinstimmungen identifizieren würde. Auch wenn diese Prüfmethode recht elementar erscheint, hat sie sich in der Praxis als sehr nützlich erwiesen.

Normalized-Compression-Distance-Test Als dritte Möglichkeit zur Bestimmung von Ähnlichkeiten von Abgaben wurde eine *Normalized Compression Distance* implementiert, die auf der theoretischen Kolmogorow-Komplexität bzw. der universellen Normalized Information Distance beruht, welche alle anderen berechenbaren Metriken minorisiert, und sich generell für jegliche Art von Daten eignet (auch UML-Diagramme oder sonstige Binärdaten, vgl. [Li+04]). Grundlage dieser Metrik ist die Abschätzung der gemeinsamen Information zweier Strings, für die eine gute Kompressionsfunktion (wie z. B. der Lempel-Ziv-Markow-Algorithmus) benötigt wird. Für Programmcode bzw. Texte sind die gleichen Normalisierungen bzw. Konfigurationen wie beim Levenshtein-Test möglich.

13.3 Architektur und eingesetzte Technologien

Die folgenden Unterabschnitte beschreiben den technischen Aufbau des GATE-Systems. Aus Platzgründen wird in Abschnitt 13.3.1 nur ein Teil seines Datenmodells genauer erläutert (Paket **userdata**, s. Abb. 13.3) und in Abschnitt 13.3.2 ein allgemeiner Überblick über die Systemarchitektur gegeben, ohne auf konkrete Implementierungsdetails einzugehen. Nähere Details zum Datenmodell und zur Architektur des GATE-Systems können in [Str09] nachgelesen werden.

13.3.1 Datenmodell

Das in Abbildung 13.3 dargestellte Datenmodell des GATE-Systems teilt sich in vier wesentliche, als Pakete dargestellte Bereiche auf. Der Bereich **taskdata** umfasst alle Daten zu den in GATE anzulegenden Aufgaben (siehe hierzu auch Abschnitt 13.2.1). Im Bereich **submissiondata** sind alle Daten definiert, die sich auf die Abgaben von Lösungen zu Übungsaufgaben durch Studierende beziehen. Der Bereich **lecturedata** kapselt alle veranstaltungs- oder vorlesungsbezogenen Daten bzw. Klassen. Der Bereich **userdata** umfasst alle Daten, die in GATE zu einem angemeldeten Nutzer angegeben werden können.

Ein Nutzer (User) kann Teilnehmer (Participation) einer in GATE angelegten Veranstaltung (Lecture) sein. Innerhalb der Veranstaltung hat jeder Teilnehmer eine bestimmte Rolle (ParticipationRole). Diese Rolle bestimmt, welche Funktionen ein Teilnehmer in der jeweiligen Veranstaltung ausführen kann. Die bei der Anmeldung in einer Veranstaltung automatisch vergebene Rolle *NormalParticipation* wird für Studierende, die Rolle *Tutor* für die Tutorinnen und Tutoren einer Veranstaltung und die Rolle *Advisor* typischerweise an Betreuer einer Veranstal-

tung vergeben. Unabhängig davon kann ein User in GATE zusätzlich den Status eines Super Users mit administrativen Rechten erhalten, der global für das gesamte GATE-System gilt. Einem Super User ist es z. B. als einzigem erlaubt, neue Veranstaltungen im GATE-System anzulegen.

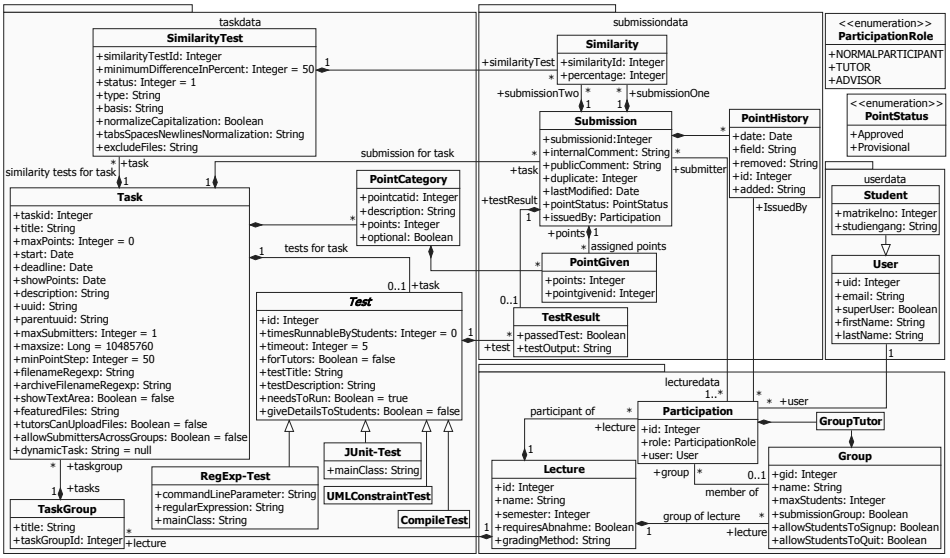

Abbildung 13.3: Das Datenmodell des GATE-Systems nach [Str09]

13.3.2 Systemarchitektur

Bei der Architektur des GATE-Systems handelt es sich um eine Vier-Schichten-Architektur, die sich in eine Benutzerschnittstelle (Graphical User Interface, GUI), die Applikationsschicht, die Persistenzschicht und die Datenbankschicht aufteilt. Grundsätzlich wurden dabei alle Schichten mit Hinblick auf Modularität und einfache Erweiterbarkeit entworfen.

Die **GUI**, die clientseitig im Webbrowser eines Nutzers läuft, nimmt Nutzereingaben entgegen und kommuniziert mittels HTTPs mit der Applikationsschicht. Die Darstellung der Views des GATE-Systems erfolgt über HTML und CSS.

Die GATE-Datenbankschicht basiert auf einem relationalen Datenbankmanagementsystem – MySQL wird dabei zurzeit als **Datenbank** eingesetzt.

Die **Persistenzschicht** enthält das in Abschnitt 13.3.1 beschriebene Datenmodell, verbirgt die Details der konkreten Datenbank und ermöglicht der Applikati-

onsschicht Zugriff auf deren Daten. Die Kommunikation mit der Datenbank wird über Java Database Connectivity (JDBC) in Verbindung mit Hibernate[2] realisiert.

Die **Applikationsschicht** kapselt die gesamte Anwendungslogik. Sie nimmt Anfragen der GUI entgegen, ändert Daten der Persistenzschicht und bereitet Daten für die GUI auf.

Um den speziellen Anforderungen einer webbasierten Anwendung gerecht zu werden, wurde für die Applikationsschicht MVC Model 2 (MVC-2), eine Variante des Model-View-Controller-Musters ([LR06], Kapitel 8.2) angewendet. Die Webserver Virtual Machine (VM) tritt dabei als Front-Controller auf, nimmt die Anfragen der Nutzer entgegen und leitet diese an spezielle Controller-Servlets weiter, welche die weitere Bearbeitung der Anfragen durchführen und schließlich an einen View bzw. weiteren Controller verweisen. Zwecks Kapselung existiert für jede Aufgabe (z. B. Abgabe einer Lösung bzw. Bewertung oder Auflistung aller Aufgaben) ein eigener Controller. Diese verarbeiten die Eingaben der Nutzer, greifen über Data-Access-Object-Klassen (DAO) der Persistenzschicht auf die Datenbank zu, führen die zugehörige Logik aus und leiten an einen entsprechenden View weiter, der schließlich HTML generiert. Den Controllern vorgelagert ist ein Servlet-Filter, der sicherstellt, dass nur authentifizierte Anfragen an die Servlets geleitet werden. Somit können weitere Funktionen hinzugefügt werden, ohne andere Controller verändern zu müssen.

Die Plagiat- und Funktionstests (Studenten- und Tutorentests) wurden innerhalb des Systems als erweiterbares Framework entworfen und implementiert, so dass relativ einfach weitere Tests bzw. Codenormalisierungen eingebunden werden können. Grundsätzlich kann so die Unterstützung durch Tests für weitere Programmiersprachen hinzugefügt werden. Einzige Voraussetzung ist, dass für das Betriebssystem des ausführenden Systems Interpreter bzw. Compiler zur Verfügung stehen und eine sichere Ausführungsumgebung eingerichtet werden kann.

Die in Abschnitt 13.2.3 beschriebenen Tests für Java-Programme werden von dem hierfür zuständigen Controller-Servlet mit Hilfe des Test-Frameworks asynchron im Hintergrund ausgeführt, so dass es zu keinen HTTP-Timeouts kommen kann. Die Ausführung kann zum einen direkt lokal in der VM des Webservers (dabei auch parallel, um alle Cores des Servers zu nutzen) oder aber z. B. durch Nutzung von Remote Method Invocation (RMI) verteilt auf mehreren Computern erfolgen. Die Plagiatstests werden nach Ablauf der Abgabefrist nicht in der Webserver VM, sondern asynchron durch den Einsatz eines Taskplaners (z. B. cron) automatisch in einer separaten VM ausgeführt, so dass Anfragen weder verlangsamt noch blockiert werden.

2 http://www.hibernate.org

Da die Funktionstests für Java-Programme automatisch die eingereichten Programme der Studierenden mit den Rechten des Webservers ausführen, wurden Sicherheitsmaßnahmen eingerichtet, um so gut wie möglich zu verhindern, dass schadhafter Code ausgeführt wird, der z. B. unberechtigterweise Dateien des Servers löscht oder Daten ändert. In diesem Zusammenhang wird die Möglichkeit der Programmiersprache Java genutzt, Rechte (sogenannte *Permissions*) für Java-Applikationen festzulegen. Über den Security-Manager kann dann verhindert werden, dass bei fehlenden Rechten bestimmte mögliche gefährliche Operationen ausgeführt werden. In GATE wird über diesen Mechanismus grundsätzlich die Ausführung aller gefährlichen Operationen (z. B. Löschen oder Schreiben von Dateien, Aufbau von Netzwerkverbindungen ins Internet) durch eine vorgegebene Policy unterbunden. Das Löschen und Schreiben von Dateien wird in einem temporären Verzeichnis für Tests erlaubt, um auch Aufgaben testen und ausführen zu können, bei denen Studierende Dateien anlegen bzw. modifizieren sollen.

Aufgrund der gewählten Systemarchitektur wird zur Ausführung von GATE eine Webserver-Software benötigt, die mit der Java-Servlet-3.0-Spezifikation kompatibel ist und einen Servlet- bzw. Webcontainer bereitstellt (z. B. Tomcat 7).

13.4 Nutzerstatistiken und Evaluationsergebnisse

GATE wird seit dem Wintersemester (WS) 2009/2010 regelmäßig im Rahmen des Übungsbetriebs mehrerer Veranstaltungen der TU Clausthal eingesetzt. Das System kommt dabei nicht nur in den in Abschnitt 13.1 genannten Einführungsveranstaltungen zur Java-Programmierung zum Einsatz, sondern auch in erster Linie zur Organisation und Verwaltung von Übungen sowie zur Unterstützung bei der Aufdeckung von Plagiaten in Einführungsveranstaltungen der Wirtschaftsinformatik und Informatik, in denen keine Java-Programmierung gelehrt wird. Tabelle 13.1 gibt eine Übersicht über die Anzahl der Nutzer, die im GATE-System der TU Clausthal seit dem WS 2009/2010 angemeldet waren.

Eine Evaluierung des GATE-Systems fand zum ersten Mal im Rahmen der Grundlagenveranstaltung zur Java-Programmierung für Studierende der Betriebswirtschaftslehre an der TU Clausthal im Jahre 2009 statt [SOP11]. Die Evaluierung ergab einen statistisch signifikanten Unterschied bezüglich der Qualität der in GATE eingereichten Lösungen der Studierenden, welche die in GATE zur Verfügung gestellten Compile-/Syntaxtests einsetzten (im Schnitt 90% der Lösungen syntaktisch korrekt) und den Lösungen der Studierenden, die dies nicht taten (im Schnitt 65% der Lösungen syntaktisch korrekt). Ebenso verhielt es sich mit den bereitgestellten Tests auf logische Korrektheit von Java-Programmen, bei denen

Lösungen mit durchgeführtem Test im Schnitt zu 52% korrekt waren und Lösungen ohne durchgeführtem Test im Schnitt zu 18%.

Bezüglich der Plagiatstests, die bei Abgaben zu Java-Programmieraufgaben durchgeführt wurden, wurde von den befragten Tutorinnen und Tutoren der Plaggie-Test als am nützlichsten empfunden (Bewertung im Schnitt 4,3 auf einer Skala von 1 bis 5, wobei 5 besser ist), der Levenshtein-Test und der Normalized-Compression-Distance-Test dagegen als weniger nützlich (durchschnittliche Bewertung jeweils 2,8 auf einer Skala von 1 bis 5).

Die befragten Tutorinnen und Tutoren bescheinigten, dass durch die zur Verfügung gestellten Funktionen des GATE-Systems der manuelle Korrektur- und Bewertungsaufwand verringert werden kann.

Einige der eingereichten Lösungen (201 Lösungen von 1031 Lösungen) wiesen nur kleinere Fehler auf (z. B. Tippfehler, Packages falsch). Die auf diese Lösungen angewendeten Syntax-/Compile-Tests bzw. Tests auf die logische Korrektheit von Java-Programmen schlugen folgerichtig fehl, da die abgegebenen Programme aufgrund der vorhandenen Fehler nicht kompilierbar waren. Die Abgaben wurden nach manueller Kontrolle der Tutorinnen und Tutoren trotzdem mit der vollen

Semester	Veranstaltungen	Betreuer	Tutoren	Studierende
SS 2016	2	4	5	164
WS 2015/2016	3	7	14	421
SS 2015	2	6	5	218
WS 2014/2015	2	3	13	399
SS 2014	2	6	7	185
WS 2013/2014	2	3	15	342
SS 2013	3	6	6	107
WS 2012/2013	2	4	18	376
SS 2012	3	7	1	113
WS 2011/2012	2	5	18	521
SS 2011	2	4	3	87
WS 2010/2011	1	2	12	341
SS 2010	1	2	2	45
WS 2009/2010	1	2	12	317
Gesamt	28	61	131	3636
Durchschnitt	2	4	9	259

Tabelle 13.1: GATE-System der TU Clausthal: Anzahl angemeldeter Nutzer pro Semester

Punktzahl bewertet. Die Tutorinnen und Tutoren haben also, wie für das System vorgesehen, eine ganzheitliche Bewertung vorgenommen, die nicht nur auf den Ergebnissen automatisiert ausführbarer Tests basiert.

Die Plagiatstests wurden insbesondere als Indikator benutzt, um mögliche Plagiate leichter erkennen zu können. Abgaben, die in GATE als mögliche Plagiate angezeigt wurden, wurden dennoch normal bewertet, wenn eine persönliche Abnahme der jeweiligen Lösung eines Studierenden durch Tutorinnen und Tutoren erfolgreich war. Das Ziel war es also herauszufinden, ob Studierende ihre eingereichte Lösung verstanden haben und erklären konnten.

Die Wirksamkeit des in GATE integrierten UML-Vergleichstests wurde in einer Studie mit 30 Teilnehmern mit geringen Programmier- bzw. Modellierungskenntnissen evaluiert [Sch+12]. Es konnte dabei festgestellt werden, dass die mit Hilfe der zur Verfügung gestellten UML-Vergleichstests erstellten Lösungen im Schnitt qualitativ besser waren (im Schnitt 94,4 von 104 Punkten), als die Lösungen, die mit dem Tool Argo-UML ohne weitere Hilfestellung generiert wurden (im Schnitt 82,2 von 104 Punkten). Die Teilnehmer, die auf Feedback eines Tutors zurückgreifen konnten, schnitten bei dem Test am besten ab (im Schnitt 100,9 von 104 Punkten). Die Qualitätsunterschiede zwischen den Lösungen der verschiedenen Gruppen waren hierbei jedoch nicht statistisch signifikant. Insbesondere konnte somit die Hypothese nicht bestätigt werden, dass durch den Einsatz des erweiterten GATE-Systems Modellierungsfähigkeiten statistisch signifikant besser erlernt werden können als durch die Verwendung eines UML-Tools ohne Feedbackfunktion. Dennoch bestätigten die Studienteilnehmer, dass durch die vom GATE-System bereitgestellten UML-Vergleichstests ein guter Lerneffekt erzielt werden kann (Bewertung auf einer Skala von 0 bis 5, wobei 5 besser ist, im Schnitt 4,4). Das durch die Tests automatisiert generierte Feedback wurde ebenfalls als sehr nützlich empfunden (Bewertung auf einer Skala von 0 bis 5 im Schnitt 4,3).

13.5 Zusammenfassung und Ausblick

Dieses Kapitel stellt das webbasierte, plattformunabhängige System GATE vor, das seit dem WS 2009/2010 regelmäßig in mehreren Veranstaltungen der TU Clausthal zum Einsatz kommt. Bei GATE handelt es sich um ein Online-Abgabesystem mit integrierter LMS-Funktionalität, das verschiedene Funktionen zur Organisation und Verwaltung von Praktika bzw. vorlesungsbegleitenden Übungen bereitstellt. Der Funktionsumfang des Systems umfasst des Weiteren eine konfigurierbare automatische Plagiatserkennung sowie verschiedene Methoden zur automatisierten Überprüfung von Java-Programmen und UML-Klassen- bzw. Akti-

vitätsdiagrammen, die Studierenden sowie Tutorinnen und Tutoren automatisiert generiertes Feedback für eingereichte Java-Programme bzw. UML-Klassen- und Aktivitätsdiagramme zur Verfügung stellen können. Zudem wird prototypisch das Anlegen bestimmter Aufgaben mit randomisierten Werten unterstützt.

Die modulare Architektur des Systems erlaubt es grundsätzlich, dieses z. B. um neue Tests für weitere Programmier- oder Modellierungssprachen zu erweitern und das bereitgestellte Feedback zu optimieren bzw. zu flexibilisieren. Ebenso können zur Optimierung der Plagiatsprüfungsfunktion weitere Algorithmen bzw. Frameworks zur Erkennung von Plagiaten integriert werden.

Das in der Programmiersprache Java geschriebene GATE-System ist eine GPLv3 lizenzierte Open-Source Software. Der aktuelle Quellcode des Systems inklusive kurzer Installationsanleitung wird auf GitHub zur Verfügung gestellt[3].

Literatur für dieses Kapitel

[AM05] Kirsti M. Ala-Mutka. „A Survey of Automated Assessment Approaches for Programming Assignments“. In: *Computer Science Education* 15.2 (2005), S. 83–102. DOI: 10.1080/08993400500150747.

[ASR06] Aleksi Ahtiainen, Sami Surakka und Mikko Rahikainen. „Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises“. In: *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*. Baltic Sea '06. ACM, 2006, S. 141–142. DOI: 10.1145/1315803.1315831.

[Bak+05] Ryan Shaun Baker u. a. „Do Performance Goals Lead Students to Game the System?“ In: *Artificial Intelligence in Education: Supporting Learning Through Intelligent and Socially Informed Technology*. Frontiers in artificial intelligence and applications 125. IOS Press, 2005, S. 57–64.

[Bak+08] Ryan Baker u. a. „Why students engage in “gaming the system” behavior in interactive learning environments“. In: *Journal of Interactive Learning Research (JILR)* 19.2 (2008), S. 185–224.

[LC04] Thomas Lancaster und Fintan Culwin. „A comparison of source code plagiarism detection engines“. In: *Computer Science Education* 14.2 (2004), S. 101–112.

3 https://github.com/csware/si, https://repo.cses.informatik.hu-berlin.de/gitlab/gate/

[Lev66] Vladimir I. Levenshtein. „Binary Codes Capable of Correcting Deletions, Insertions and Reversals“. In: *Soviet Physics Doklady* 10 (1966), S. 707.

[Li+04] Ming Li u. a. „The similarity metric“. In: *Information Theory, IEEE Transactions on* 50.12 (2004), S. 3250–3264.

[LR06] Bernhard Lahres und Gregor Raýman. *Praxisbuch Objektorientierung – Professionelle Entwurfsverfahren*. Galileo Computing, 2006. ISBN: 978-3-88579-282-6. URL: http://openbook.galileodesign.de/oo/.

[MS13] Oliver Müller und Sven Strickroth. „GATE – Ein System zur Verbesserung der Programmierausbildung und zur Unterstützung von Tutoren“. In: *Workshop „Automatische Bewertung von Programmieraufgaben“ (ABP 2013)*. Bd. 1067. CEUR Workshop Proceedings. 2013.

[PMP00] Lutz Prechelt, Guido Malpohl und Michael Philippsen. „Finding Plagiarisms among a Set of Programs with JPlag“. In: *JOURNAL OF UNIVERSAL COMPUTER SCIENCE* 8 (2000), S. 1016–1038.

[Sch+12] Joachim Schramm u. a. „Teaching UML Skills to Novice Programmers Using a Sample Solution Based Intelligent Tutoring System“. In: *Proceedings of the 25th International Conference of the Florida Artificial Intelligence Research Society (FLAIRS)*. Marco Island, FL, USA: AAAI, 2012, S. 472 –477.

[SOP11] Sven Strickroth, Hannes Olivier und Niels Pinkwart. „Das GATE-System: Qualitätssteigerung durch Selbsttests für Studenten bei der Onlineabgabe von Übungsaufgaben?“ In: *DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik*. Bd. 188. LNI. GI, 2011, S. 115–126.

[Str09] Sven Strickroth. *Unterstützungsverfahren für Tutoren bei der Online-Abgabe von Übungsaufgaben*. Bachelorarbeit. 2009.