Spezifikation zum Software Praktikum "JUnit Backend"

Christian Baumann und Julia Preusse

21. November 2007

Inhaltsverzeichnis

1 Aufgabenstellung:

Implementieren Sie ein Backend, das Lösungen von Java-Aufgaben auf syntaktische Korrektheit und mit Hilfe durch den Prüfer hintertlegten Unit-Tests auf semantische Richtigkeit hin überprüft.

1.1 Vorgehensweise bei der Aufgabenbearbeitung:

Zunächst wird sich unser Team in die von uns gewählte Korrektheitsüberprüfungsmethode (JUnit-Tests) einarbeiten, deren Funktionsweise verstehen und nachvollziehen können. Des Weiteren muss die Programmiersprache Python erlernt werden, um den Aufbau der bestehenden Komponenten, wie dem ECSpooler und anderen Backends, nachzuvollziehen und eine nahtlose Integration gewährleisten zu können.

1.2 Erwartete Ergebnisse:

Neben dem Erlernen einer neuen Programmiersprache (Python) und dem Prinzip der JUnit-Tests ist es für uns interessant uns in ein bereits existierendes Projekt einzuarbeiten und unsere Teamfähigkeit zu schulen.

1.3 Meilensteine (Projektplan):

2007-07-06	Einreichung der Spezifikation
2007-07-26	Treffen mit Frau Blümel und Herrn Amelung
2007-07-31	Einarbeitung in bestehende Backends
	Einarbeitung in JUnit-Tests
	Beginn der Implementierung
Mitte 2007-08	Testen des Backends
Ende 2007-08	Ausarbeitung der Präsentation
	Ausarbeitung des Handbuchs
2007-10-08	Präsentation des Backends

2 Vorbetrachtungen:

2.1 Zusammenfassung:

Die Studierenden sollen ein Programm in Java schreiben, das eine vom Prüfenden festgelegte Funktionalität erfüllen soll. Name der auszuführenden Methode, Argumente und Typ des Rückgabewerts werden in der Aufgabenstellung vorgegeben. Die vom Prüfenden hinterlegten JUnit-Tests werden dazu verwendet, um auf die gewünschte Funktionalität hin zu prüfen.

2.2 Allgemeine Eigenschaften:

2.2.1 Eingabe:

Der Studierende reicht den Programmcode als Text(datei) ein, wobei das Programm entweder im Textfeld editiert oder hochgeladen werden kann.

2.2.2 Art des Tests:

Das eingereichte Programm wird zunächst syntaktisch überprüft und danach mittels Unit-Tests auf die korrekte Funktionsfähigkeit hin getestet.

2.2.3 Syntaxüberprüfung:

Die Syntaxüberprüfung wird durch den Java-Compiler durchgeführt, da dessen Fehlermeldungen bereits so ausgereift und verständlich sind dass man keine weiteren Zusätze zur Ausgabe braucht. Das eingereichte Programm wird ausgeführt und die gegebenenfalls auftretetenden Fehler werden ausgegeben.

2.2.4 Testdaten:

Der Lehrende muss geeignete Unit-Tests hinterlegen, sodass die Einreichung auf semantische Korrektheit geprüft werden kann. Die vom Unit-Test generierten Ausgaben werden dem Studierenden ausgegeben.

2.2.5 Ergebnis der Überprüfung:

Das Backend gibt nur dann ein positives Feedback zurück, wenn das Programm syntaktisch korrekt ist und alle Unit-Tests fehlerfrei ausgeführt wurden. Sobald ein Test fehlschlägt, wird ein negatives Feedback zurückgegeben und die entsprechende Fehlermeldung an den Studierenden ausgegeben.

3 Aufgabensteller:

Dem Aufgabensteller obliegen einige Pflichten bei dem Entwurf der Aufgabenstellung, wenn die studentische Lösung überprüft werden soll. Diese werden im folgenden beschrieben:

3.1 Informationen für den Studierenden:

Zunächst einmal sollte der Lehrende den Studierenden die von ihm verwendete Java-Version mitteilen. Der Aufgabenstellende muss dem Studierenden genau vorgeben, wie er die Methoden zu benennen hat, damit ein einwandfreier Ablauf des Testprogrammes gewährleistet ist. Des Weiteren muss die gesamte Methodensignatur textuell oder als Antwortvorlage vorgegeben werden. Außerdem muss die Funktionalität der zu implementierenden Methoden genau beschrieben werden.

3.2 Informationen für das Backend:

Der Aufgabensteller muss die Testdaten in Form von korrekten Unit-Test-Methoden (Syntax der JUnit Version 4) eingeben. Zudem kann der Aufgabensteller Importe und Hilfsfunktionen angeben und hat die Möglichkeit in den Unit-Tests die Klasse des Studierenden mit "${CLASS}" zu referenzieren.

4 Student:

4.1 Art / Bestandteile der Einreichung:

Der Studierende löst die Anforderungen der Aufgabenstellung gemäß und reicht seine unkompilierte Programmklasse in Form von plain text ein.

4.2 Rückmeldung zur Einreichung:

Der Studierende bekommt gemäß seiner Aufgabenbearbeitung eine Rückmeldung des Systems, ob die von ihm eingereichte Lösung die erwarteten Ergebnisse geliefert hat oder nicht. Dabei wird im syntaktischen Fehlerfall die Meldung des Compilers und im semantischen Fehlerfall die des JUnit-Tests ausgegeben.

Bei den semantischen Fehlern werden dem Studierenden die geforderten und die tatsächlich erhaltenen Ausgaben angezeigt. Somit hat dieser die Möglichkeit, Fehler in seinen Methoden zu erkennen und diese entsprechend zu überarbeiten.

5 Backend:

5.1 Voraussetzungen:

Die primäre Vorraussetzung für das Backend ist ein funktionierendes Java-System. Hierfür werden wir eine Java-Version (ab Java1.5) benutzen, die Unit-Tests (Version 4) unterstützt und auf die nötigen JUnit-Bibliotheken zugreifen kann. Benötigen Methoden bestimmte Pakete (wie etwa „algds.jar“), so sind auch diese zu hinterlegen.

5.2 Funktionsweise:

Nachdem der Studierende seine Lösung eingereicht hat, wird diese Datei vom Backend entgegengenommen, der Klassenname für spätere Zwecke extrahiert und wie folgt verändert, um die Kompilierbarkeit zu gewährleisten:

- Es wird eine neue package-Deklaration geschrieben, die das Verzeichnis enthält, in das die Klasse vom Backend aus geschrieben wird. Besitzt die Klasse bereits eine package-Deklaration, so wird diese durch die neue ersetzt.
- Es werden alle Importe überprüft:
  - Importe die im package "java.*"liegen bleiben unverändert.
  - Importe die nicht im package "java.*"liegen werden so verändert, dass die angegebenen Klassen im Bibliothekenordner (package junit_libs) gesucht werden.

Beim syntaktischen Korrektheitstest wird diese veränderte Datei lokal mit der Endung „.java“ gespeichert, kompiliert und die dabei eventuell anfallenden Fehlermeldungen des Compilers gesammelt und dem Studierenden ausgegeben. Bei aufgetretenen Fehlern wird ein negatives BackendResult zurückgegeben.

Für die semantische Überprüfung ist das Wrapper Template zuständig, das vor der Ausführung den folgenden Schritten unterzogen wird:

- Durch den Lehrenden angegebene import-Anweisungen werden in das Wrapper Template geschrieben.
- Dem Wrapper Template wird der Pfad der studentischen Klasse hinzugefügt, so dass diese importiert wird.
- Die gegebenenfalls definierten Hilfsfunktionen und die Unit-Tests des Lehrenden werden in das Wrapper Template geschrieben, damit diese für einander sichtbar sind und vom Wrapper Template aufgerufen werden können.
- Die definierte Variable "${CLASS}" wird durch den Klassennamen der studentischen Einreichung ersetzt.

Das so veränderte Wrapper Template wird nun lokal als Javadatei gespeichert und kompiliert. Eventuelle Fehler sind auf den Quelltext des Wrapper Templates zurückzuführen und werden deshalb nicht dem Studierenden zurückgegeben. Verlief das Kompilieren fehlerfrei, wird die kompilierte Datei dem Interpreter übergeben, so dass dieser das Programm ausführt. Das Wrapper Template ist so aufgebaut, dass alle spezifizierten Unit-Tests durchlaufen und deren Ausgaben als von Java bereitgestelltes „Result“ gespeichert werden. Traten semantische Fehler während des Durchlaufs auf, so wird der erste im "Result"gespeicherte Fehler ausgegeben, ein negatives Feedback zurückgegeben und das Programm mit Exitcode 1 terminiert. Traten keine semantischen Fehler auf, so terminiert das Programm mit Exitcode 0 und dem Studierenden wird eine positive Meldung ausgegeben.

6 Tests:

Um die Funktionstüchtigkeit des Backends zu garantieren, wird es folgenden drei Arten von Tests unterzogen:

- Test einer syntaktisch nicht korrekten Einreichung
- Test einer semantisch nicht korrekten Einreichung
- Test einer fehlerfreien Einreichung

Ein vollständiges Testbeispiel wird ausführlich im Handbuch beschrieben.

7 Sicherheit:

Die Sicherheit wird duch bestehende Strukturen des Servers erzielt. So läuft auf dem Server ein NetBSD, das mit „systrace“ unautorisierte Funktionsausführungen unterbindet. Um Endlosrekursionen vorzubeugen ist jedes Programm an eine Zeitschranke gebunden. Wenn das Programm nicht innerhalb der eingestellten Zeitschranke terminiert, wird der Prozess mit einem „kill“-Signal beendet.

8 Probleme:

Ein beobachtetes Problem, das mit der Nutzung des Backends auftreten kann, ist folgendes:

Existiert in einer studentischen Einreichung mindestens eine der geforderten und im Unit-Test verwendeten Methoden nicht (beispielsweise wegen eines anderen Methodennamens), so gibt der Java-Interpreter die Stellen des Wrapper-Codes aus, an welchen Fehler aufgetreten sind. Das führt dazu, dass dem Studierenden die Unit-Tests ausgegeben werden könnten.

Behoben wurde das Problem durch einen Regulären Ausdruck, der die Ausgabe des Interpreters so beschneidet, dass dem Studierenden zwar noch die Meldung ausgegeben wird welche Methode nicht gefunden wurde, nicht aber den kompletten Unit-Test.

Bislang erkannten wir noch keine bessere Lösung als diese, wobei das Problem an sich ein generelles zu sein scheint, da auch andere Backends mit diesem Problem Schwierigkeiten haben.