Der Funktions-Fragment-Checker: eine effektive Übungsumgebung für C# -Programmieranfänger

Matthias Längrich, Antje Meyer, Jörg Schulze

Fachbereich Informatik
Hochschule Zittau/Görlitz (FH)
Brückenstrasse 1
02828 Görlitz
mlaengrich@hs-zigr.de, meyer05@stud.hs-zigr.de, joerg.schulze@hs-zigr.de

Abstract: Mit dem im Task Trainer (TT) integrierten Funktions-Fragment-Checker (FFC) wurde für die C#-Programmierausbildung eine computergestützte Übungsumgebung geschaffen, in der der Lernende durch die eigenständige Auswahl von unterschiedlich komplexen Übungsaufgaben aus einem Pool verschiedener Aufgabentypen seinen Lern- und Übungsprozess aktiv und selbstgesteuert gestalten kann. Unter Üben wird das Konstruieren von Lösungswegen durch den Lernenden verstanden. Gleichzeitig erhält dieser einen individuell bestimmbaren Einblick in die Syntax und Semantik der Programmiersprache als Vorraussetzung eines besseren Verständnisses informatischer Konzepte.

1 Den Einstieg ins Programmieren mit dem FFC leichter gemacht

Das Grundwissen um die Konzepte der Informatik ist Teil der Allgemeinbildung.[1] Eine wichtige Rolle spielt dabei das Erlernen einer Programmiersprache, was mittlerweile ein fester Bestandteil des Lehrplans der gymnasialen Oberstufe in Sachsen[2] und der Curricula vieler Studiengänge an Universitäten, Fachhochschulen, Berufsakademien, sowie bei Umschulungen und Weiterbildungen ist.

Insbesondere in der Grundlagenausbildung soll der hier vorgestellte FFC eine unterstützende Rolle übernehmen. Er hilft dem Lernenden von der komplexen Syntax und Semantik einer Programmiersprache wie C# zu abstrahieren. In den vom FFC bereitgestellten Übungsaufgaben programmiert der Lernende anfangs imperativ und geht erst später bzw. mit dem entsprechenden Vorwissen zum objektorientierten Programmieren über. Diese Herangehensweise an das Erlernen einer Programmiersprache zielt insbesondere auf den Lernenden ohne oder nur mit geringer Programmiererfahrung ab.

[1] [Br94]
[2] [SBS05]

2 Anforderungen an eine Programmierausbildung für Anfänger

Nach Böszörményi muss eine Programmierausbildung grundlegende Programmierkonzepte so vermitteln, dass sie vom Lernenden als Basis für fortgeschrittene Konzepte und Programmdarstellungen verwendet werden können. Zu Beginn der Programmierausbildung soll auf die Objektorientierung verzichtet werden, da sie den Lernenden für längere Zeit eher daran hindert, die Grundlagen richtig zu verstehen.[3]

Gerade beim Erlernen einer Programmiersprache wie C# treten anfangs bestimmte Schwierigkeiten hinsichtlich der Syntax und Semantik auf. Mit der Semantik ist das der Programmiersprache zugrunde liegende Framework gemeint, das dem Softwareentwickler eine reiche Funktionalität bietet, aber für den Anfänger eher verwirrend als hilfreich ist. Die Syntax von C# stellt sich ebenso komplex dar. So wird beispielsweise für das Testen immer ein komplettes Programm benötigt, denn Programmfragmente allein können nicht übersetzt und demzufolge auch nicht getestet werden. Für die Grundlagenausbildung sollte jedoch eine einfache Syntax und Semantik bevorzugt werden.[4]

Auf die Bedeutung einer geeigneten Entwicklungsumgebung für Programmieranfänger, welche die Konzentration auf die Programmlogik erleichtert, weist Reichert hin.[5]

Darüber hinaus ist die Verifizierung der Lösungen von Übungsaufgaben[6] ein weiterer zu beachtender Aspekt. Besteht die konkrete Aufgabe für einen Lernenden zum Beispiel darin, eine Funktion für eine bestimmte Problemstellung zu implementieren, so möchte er natürlich auch anschließend wissen, ob seine Lösung korrekt ist.

3 Die Verwendbarkeit von Programmier-Paradigmen

Während das objektorientierte Paradigma sich für mittlere bis große Projekte eignet, steht am Anfang der Programmierausbildung das „Programmieren im Kleinen“ im Sinne des imperativen Paradigma.

Durch das Üben von zunächst einfachen Ausdrücken, Anweisungen, Funktionsfragmenten (Folge von Anweisungen ohne Deklarationen und Initialisierungen) sowie Funktionen findet der Programmieranfänger einen leichteren Zugang zu einer komplexen Sprache wie C#. Denn durch das Programmieren von Fragmenten entlastet sich der Lernende zunächst von einer ihm anfangs noch nicht überschaubaren Syntax bzw. Semantik. Diese Kapselung bzw. das Zurückhalten des syntaktischen bzw. semantischen Aspekts löst der Lernende selbst auf, indem er durch sein wachsendes Wissen immer komplexer werdende Fragmente bis hin zu kompletten Programmen entwickelt.

---

[3] [Bö01]
[4] [Li02]
[5] [RNH04]
[6] [KSZ02]

Bereits für das Programmieren von einfachen Fragmenten können Übungsaufgaben formuliert werden, die vom Lernenden gelöst und anschließend getestet werden können. Für das Üben von C# werden generell zwei verschiedene Aufgabentypen mit jeweils unterschiedlichem Komplexitäts- und Schwierigkeitsgrad von einander differenziert.

In den so genannten TYP-1-Aufgaben kann das Ergebnis, z.B. ein Ausdruck, konkret geprüft werden.

Eine TYP-2-Aufgabe für Funktionsfragmente bzw. Funktionen muss dagegen einer vorher gegebenen Spezifikation genügen und lässt daher unterschiedliche Lösungen zu. Das Verifizieren der Lösungen ist umständlich, da im Allgemeinen ein Black-Box-Test verwendet wird.

Der im Folgenden näher vorgestellte TT und sein Modul des FFC überträgt das vom Extreme-Programming bekannte Verfahren des Unit-Tests[7] auf Ausdrücke, Anweisungen, Funktionsfragmente und Funktionen und testet mit vorgegebenen Daten. Ein Vorteil dabei ist, dass der Lernende diese Testfälle nicht kennt.

4 Der FFC als ein Modul des TT und technischer Hintergrund

Da der FFC nur einen Teil des TT darstellt, soll dieser kurz vorgestellt werden.

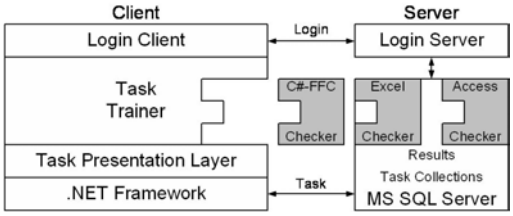

Abbildung 1: Der Task Trainer

Wie in der Abbildung 1 zu sehen, stellt der TT eine klassische Client-Server-Applikation dar. Sobald der Lernende eine Aufgabe im TT ausgewählt hat, werden die passende Check-Engine, das Aufgabendokument, der Lösungsstand und die Vergleichsdokumente vom Server heruntergeladen. Der eigentliche Check der Lösungen erfolgt dann ausschließlich clientseitig, was erheblich zur Leistung des Systems beiträgt.

Abbildung 2 zeigt einen Screenshot des FFC innerhalb des TT. Die Nummern kennzeichnen die entsprechenden Bereiche, die im Folgenden erklärt werden.

[7] [BK03]

Insgesamt drei **Aufgabenansichten** (1) zeigen alle verfügbaren[8], empfohlenen und bereits angefangenen bzw. beendeten Aufgaben an.

In dem **Aufgabendokument** (2) sind die Übungsaufgaben beschrieben. Ersichtlich sind die einzelnen Checkpunkte, welche sich nach einer Teilaufgabe anschließen und den Zeitpunkt der Verifizierung der Lösung zeigen. In dem **Editor** (3) entwickelt der Lernende seine Lösung. Die drei **Checkboxen** (4) werden durch die Check-Engine gesetzt und zeigen an, ob es beim Testen ein syntaktisches Problem, einen Timeout, ein Problem beim Black-Box-Test oder kein Problem gegeben hat.

Mit dem Betätigen des **Checkbuttons** (5) wird der Check-Vorgang ausgelöst. Die zur ausgewählten Aufgabe gehörenden **Checkpunkte** (6) werden in einer Liste angezeigt. Der jeweils zu testende Checkpunkt muss dabei aktiviert werden.[9]

Nachdem ein Check durchgeführt wurde, wird in der **Check-Ausgabe** (7) der Check-Engine angezeigt, inwieweit der Checkpunkt der aktuellen Aufgabe bereits korrekt abgeschlossen wurde. Syntaktische Fehler werden als Compilermeldung in der **Compiler-Ausgabe** (8) ausgegeben.

Die **Statuszeile** (9) zeigt die Position der Aufgabe in der Aufgabensammlung. (Sprache, Aufgabensammlung, Kapitel, Abschnitt, Aufgabennummer).

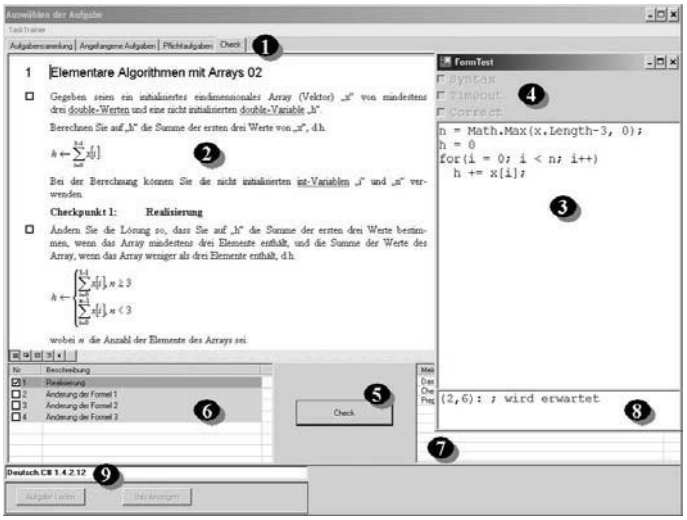

Abbildung 2: Der Funktions-Fragment-Checker

Wie in Abbildung 2 zu sehen, reicht es aus, Funktionsfragmente zu entwickeln. Dafür wird dem Lernenden zu Beginn eine Reihe von Variablen bzw. Objekten zur Verfügung gestellt, ohne dass er diese deklarieren bzw. initialisieren muss.

---

[8] Sie sind in Aufgabensammlungen angeordnet, aus denen der Betreuer die empfohlenen Aufgaben frei auswählen kann.

[9] Der Lernende kann jederzeit die Lösung des letzten korrekten Checkpunktes wiederherstellen.

Dem Lernenden wird es überlassen, wie er die Spezifikation erfüllt. Wird die dreistufige Verifizierung bestanden, so ist der Checkpunkt abgeschlossen. Sind alle Checkpunkte korrekt, gilt die Aufgabe als gelöst.

Falls als Lösung ein bestimmter Algorithmus entwickelt werden soll, so kann dies mit Hilfe verschiedener Bedingungen (z.B. dem Verbot gewisser Operationen) gesteuert werden. Die Aufgaben sind zudem wiederholbar, bis der Lernende mit sich zufrieden ist.

5 Didaktischer Hintergrund

Die Übungsaufgaben als Hauptbestandteil der durch den FFC gegebenen Übungsumgebung orientieren sich stark am konstruktivistischen Lerngedanken. Demnach ist zufolge neuerer Lernkonzeptionen das Lernen (Üben) ein selbstständig zu vollziehender Prozess mit starker Situationsbindung, in dessen Verlauf Wissen, Inhalte, Fähigkeiten etc. nicht aufgenommen, sondern konstruiert werden. Dieser Konstruktionsprozess beginnt jedoch niemals bei Null, sondern hat als Basis immer die vorhandene Wissensstruktur. Vorwissen ist immer der Ausgangspunkt für die Interpretation von Informationen, die zu Lernen als Konstruktion von Wissen führen kann. Ein solches Lernen ist wiederum stark kontext- und situationsabhängig. Der Lernende selbst kann sich diesen Prozess vergegenwärtigen und hat auf meta-kognitiver Ebene Vorstellungen darüber, wie er lernt, unter welchen Bedingungen er am besten lernt und wie er sein Lernen organisieren kann.[10]

6 Effektives Üben mit dem FFC

Der einfache Umgang und das leichte Zurechtfinden im FFC als auch das Verifizieren der Lösungen machen das Üben sowohl innerhalb als auch außerhalb einer Übungseinheit bzw. eines Seminars für den Lernenden attraktiv. Der Lernende muss nicht erst auf die Rückmeldung des Betreuers über die Richtigkeit seines Ergebnisses warten, sondern kann sich selbst überprüfen. Darüber hinaus ermöglicht der FFC dem Lernenden seine Übungszeiten und -wege individuell zu bestimmen. Der Übungs- und somit auch Lernprozess wird von dem Lernenden komplett und aktiv selbst gesteuert.

Die individuelle Herangehensweise an den Übungsaufgaben bzw. an das Entwickeln von Lösungsmöglichkeiten und -wegen macht deutlich, dass bei jedem Lernenden andere Probleme bei der Bewältigung einer Aufgaben auftreten können. Somit ist jeder auf unterschiedlichster Weise auf die Hilfe des Betreuers angewiesen.

Durch das selbständige Testen der Ergebnisse durch den Lernenden bleibt auch dem Betreuer ein hoher Zeitaufwand für das Überprüfen und Korrigieren von Lösungsaufgaben erspart und er wird dadurch von Routineaufgaben befreit. Außerdem kann er so besser auf die individuellen Bedürfnisse des Lernenden eingehen bzw. dem Lernenden individuelle Rückmeldung über seine Leistung geben.

---

[10] [Te02]

Sowohl der Lernende wie auch der Lehrende ziehen ihren Nutzen aus dieser Applikation, welche die Effektivität und Effizienz von Übungen dieses Lehrgebiets signifikant verbessert

7 Ausblick

Zukünftig wird eine weitere Check-Engine an das Visual Studio .NET gekoppelt, um bei größeren Projekten diese IDE einzusetzen und zu schulen. Damit wird der FFC in der C#-Programmiergrundausbildung auf geeigneter Weise ergänzt.

Wie in Abbildung 1 zu sehen, existieren neben dem FFC weitere Check-Engines für den TT. In naher Zukunft sollen diese und auch der FFC sowohl technisch als auch inhaltlich ausreifen und an die Bedürfnisse der Lernenden und Lehrenden angepasst werden. Zu diesem Zweck soll eine qualitative Evaluation genutzt werden.

Literatur

[BK03] Barnes, D. J.; Kölling, M.: Objets first with Java: A Practical Introduction Using BlueJ. Prentice Hall, München, 2003; S. 143.

[Bö01] Böszörményi, L.: Java für Anfäger? Warum Java nicht meine Lieblingssprache für einen Anfängerkurs ist. LOG IN 21, 1, 2001; S. 14-15.

[Br94] Breier, N.: Informatische Bildung als Teil der Allgemeinbildung. LOG IN 14, 5/6, 1994; S. 90-93.

[KSZ02] Krinke, J.; Störzer, M.; Zeller, A.: Web-basierte Programmierpraktika mit Praktomat. Gesellschaft für Informatik e.V., Software Trends 22, 3, 2002; S. 15-30.

[Li02] http://www.icc-computer.de/ingo/diplom/Diplomarbeit.pdf; Linkweiler, I.: Eignet sich die Skriptsprache Python für schnelle Softwareenwicklungsprozess?; 8.06.2005; S. 22-30.

[RNH04] Reichert, R.; Nievergelt, J.; Hartmann, W.: Programmieren mit Kara. Springer; Berlin, 2004; S. 11.

[SBS05] http://marvin.sn.schule.de/~infogy/didaktik/lplan/lehrplan.html: Sächsischer Bildungsserver (SBS); 17.02.2005.

[Te02] Terhart E.; LSW (Hrsg.): Konstruktivismus und Unterricht. Verlag für Schule und Weiterbildung, Bönen, 2. Auflage, 2002; S. 25.