


Programmier-Praktikum für Erstsemester – Erfahrungen aus mehreren Iterationen

Markus Dahm ¹, Jennifer Rose², Marius Köhler³

Abstract: Die Wirklichkeit in der Programmier-Ausbildung von Erstsemestern hat immer wieder überraschende Erkenntnisse gebracht. Seit 2014 wurden sowohl ein didaktisches Konzept als auch eine unterstützende integrierte Entwicklungsumgebung (5Code) entwickelt, die speziell ProgrammieranfängerInnen unterstützen soll. Das didaktische Konzept vermittelt von Anfang an, wie man in fünf Schritten *vom Problem zum Programm* kommt: *Lesen* → *Verstehen* → *Überlegen* → *Aufschreiben* → *Codieren*. In bisher sechs Iterationen wurden das Konzept und 5Code aufgrund der Evaluation immer wieder angepasst und neu implementiert. Einige Erfahrungen daraus, sowie die daraus gezogenen Schlüsse für die weitere Verbesserung werden in diesem Praxisbeitrag vorgestellt.

Keywords:

Programmieren, Lernen, Anfänger, IDE, Entwickeln, Hochschule, Sekundarstufe, Software-Entwicklung, Didaktik der Informatik, Kognitive Belastung, Erfahrungen, Medienkompetenz


1 Ziel

1.1 Vom Problem zum Programm in fünf Schritten

Unser Ziel besteht darin, dass AnfängerInnen sowohl die Vorgehensweise vom Problem zum Programm begreifen, als auch, dass sie diese praktisch anwenden können.

Lernende eignen sich die Syntax und Semantik von Programmiersprachen oft schnell an, wissen dann jedoch oft nicht, wann und wie das theoretisch Gelernte auch praktisch angewendet werden soll [FV10], [Wi14], [Hu07]. Dazu gehört auch, dass sie sowohl die Aufgabe verstehen als auch Mittel und Wege kennen, wie sie zu einer Lösung kommen.

Übliche Entwicklungsumgebungen bieten da kaum Hilfestellung, da diese nur Code behandeln. Bestehende Angebote speziell für ProgrammieranfängerInnen, wie BlueJ, Greenfoot oder Scratch, bieten Unterstützungen nur beim Codieren, decken jedoch Bereiche wie Problemanalyse und Dokumentation von eigenen Ideen und Lösungsansätzen nicht ab [Da15]. Diese sind für den kompletten Ablauf grundlegend wichtig, werden aber vor allem von Anfängern selten wahrgenommen und daher unterschätzt [Hu07], [Lo16].

¹ HS Düsseldorf, FB Medien, Münsterstraße 156, 40476 Düsseldorf, markus.dahm@hs-duesseldorf.de, 
<https://orcid.org/0000-0000-0000-0000>

² HS Düsseldorf, FB Medien, Münsterstraße 156, 40476 Düsseldorf, jennifer.rose@hs-duesseldorf.de

³ HS Düsseldorf, FB Medien, Münsterstraße 156, 40476 Düsseldorf, marius.koehler@study.hs-duesseldorf.de

Daher vermittelt das von uns entwickelte didaktische Konzept von Anfang an die Vorgehensweise *vom Problem zum Programm* in für Anfänger verständlichen fünf Schritten:

Lesen → *Verstehen* → *Überlegen* → *Aufschreiben* → *Codieren* kurz: *LVÜAC*

LVÜAC betont vor allem die ersten, vorbereitenden Schritte. Codieren wird dabei bewusst als letzter Schritt dieser Kette dargestellt, da es sehr häufig für den einzigen Schritt bei der Software-Entwicklung gehalten wird. Natürlich gehört auch das Testen zur Software-Entwicklung, es steht aber vor allem in den ersten beiden Semestern nicht im Vordergrund.

1.2 Reduktion der kognitiven Belastung

Die kognitive Belastung nach der Cognitive Load Theory von Sweller und Chandler [CS96] bei der Lösung von Aufgaben ist für Programmieranfänger sehr hoch [Wi14]: Im kapazitiv sehr begrenzten Arbeitsgedächtnis, das den kognitiven Kontext umfasst, müssen gleichzeitig 1) die Aufgabenstellung, 2) das Verständnis der Aufgabe, 3) die semantischen und 4) syntaktischen Mittel der Programmiersprache gehalten werden. In diesem Kontext soll 5) ein Lösungskonzept entwickelt werden und dann auch noch 6) fehlerfrei und richtig formatiert eingetippt und dabei 7) eine (komplexe) IDE genutzt werden. Die Reduktion dieser kognitiven Belastung ist daher ein wesentlicher Ansatz, um den Lernerfolg von ProgrammieranfängerInnen zu verbessern.

2 Iterationen im praktischen Einsatz

Um die kognitive Belastung der Lernenden wirksam zu vermindern und so den Lernerfolg zu erhöhen, wurde die IDE 5Code entwickelt, die über alle fünf Schritte den gesamten Kontext integriert; von der Aufgabenstellung über eigene Notizen bis zur Codierung. Inhalte können dabei über alle Dokumente miteinander verknüpft werden [Da15], [Da16]. Das didaktische Konzept und 5Code wurde ab dem Wintersemester 2014/15 im Praktikum Objektorientierte Programmierung 1 im Studiengang B.Sc. Medieninformatik am Fachbereich Medien der Hochschule Düsseldorf mit 86 bis 104 Studierenden eingesetzt.

In jedem Semester wurden zweimal evaluiert (Mitte und Ende des Semesters mittels eines Online-Fragebogens) sowie das Verhalten im Praktikum beobachtet und dokumentiert. Als Konsequenz wurden Konzepte und Umsetzungen angepasst, verworfen oder neu hinzugefügt. Die Vorgehensweise entspricht damit einem Design-Based Research [An12].

Zu Beginn (2014 bis 2016) wurde eine sehr strikte Struktur der Vorgehensweise vom Problem zum Programm vorgegeben: Die Aufgabe steht in einer eigenen Spalte links, dort kann Text markiert werden, der als Notiz in einer eigenen Spalte daneben organisiert werden kann. Aus jeder Notiz wird automatisch in der Code-Spalte ein Kommentar, der mit eigenem Code ergänzt werden soll. Diese Struktur bildet die Phasen LVÜAC so sehr klar in unterschiedliche, streng voneinander getrennte Artefakte ab (Abb.1).

2 Das kleine Einmaleins

a) Geben Sie das **Einmaleins von 1 bis n** auf die Konsole in Form einer **Tabelle** aus. Die Obergrenze n soll dabei variabel sein (d.h. Sie definieren dafür eine Variable).
Verwenden Sie dafür verschachtelte for-Schleifen. Dabei definieren **anstelle** Sie die Schleifenvariablen im **ersten** umgesetzten Bereich im

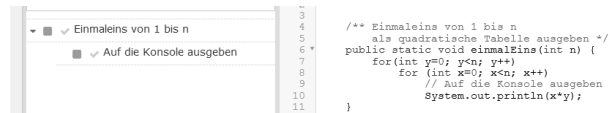


Abb. 1: 5Code – 2014-16 – Aufgabe mit Markierungen, Eigene Notizen, Code mit Kommentar

Das Feedback der Studierenden führte in den Jahren 2017 bis 2018 zur Version *5Code.Local* [Ki17]: Um die eigenen Überlegungen mit dem Code stärker zu integrieren, haben wir *Flow-Programming* als eine aktualisierte Version des „Literate-Programming“ [Kn84] entwickelt [Ro20]. Die Artefakte aller Phasen (Notizen und Code) werden im Fluss (*Flow*), d.h. in der Reihenfolge ihrer Bearbeitung in einem einzigen Dokument integriert. Lesen, Schreiben, Codieren und Orientieren geschieht so im gleichen Kontext, die kognitive Belastung wird so verringert. Außerdem können Code-Blöcke ohne den in Java nötigen Klassen-Code sofort ausgeführt werden. Das vereinfacht den Umgang mit einfachen prozeduralen Programmstrukturen und vereinfacht das Explorieren und Testen (Abb.2).

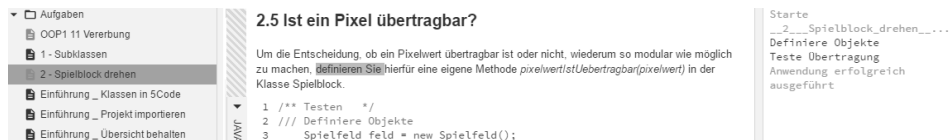


Abb. 2: 5Code.Local - 2017-18: Projekt-Navigation, Aufgabe mit Test-Code-Block, Klassen-Code

Das Konzept *Flow-Programming* wurde im Praktikum positiv evaluiert, allerdings zeigte sich, dass dieses neuartige Konzept einer besonderen Unterstützung bei der Navigation über mehrere Dokumente (Aufgaben und Lösungen) hinweg bedarf (s. 4.2). Daher wurde 2019 die Version *5CodeTNG* neu konzipiert und implementiert [Kö19], [Ro20] (Abb.3).



Abb. 3: 5CodeTNG - 2019: Code-Explorer, Aufgabe mit Test-Code-Block, Klassen-Code

Die Navigation sowohl über Dokumente als auch über den Code wurde vereinfacht und verbessert. Aufgaben und Lösungen werden in zwei Dokumenten angelegt, die beim Scrollen gegenseitig synchronisiert werden („SynchScroll“). Das vereinfacht insbesondere die Arbeit bei Objekt-orientierten Themen stark.

3 Erfolgreiche Konzepte

Die folgenden innovativen Konzepte haben sich im Verlaufe der Iterationen bewährt, wie die Evaluationen zeigen, sie wurden jeweils neu entwickelt oder stetig weiter verbessert.

Das Konzept *LVÜAC - Lesen → Verstehen → Überlegen → Aufschreiben → Codieren* hat sich als sinnvoll und nutzbringend über alle hier beschriebenen Iterationen erwiesen. Zunächst wird das Aufschreiben vor dem „eigentlichen“ Codieren als lästig empfunden, gegen Ende des Semesters haben die meisten Studierenden aber den Nutzen erkannt. Vor allem bei Fragen nach Hilfe im Praktikum wurde ihnen deutlich, dass die Beantwortung vereinfacht wird, wenn sie ihr Verständnis der Aufgabe und ihre eigenen Überlegungen kurz aufgeschrieben hatten und nicht nur falschen Code vorweisen konnten.

Die *Integration aller Dokumente über alle Phasen*, von den Aufgaben über eigene Notizen bis hin zu eigenem Code wurde als sehr hilfreich wahrgenommen und auch gerne genutzt.

Flow-Programming wurde schnell verstanden und gerne genutzt. Dass Lösungs-Code direkt im Aufgabentext steht, ist bei kleinen und einfachen Aufgaben von großem Vorteil: Der Zusammenhang ist sofort ersichtlich, es muss nichts gesucht werden. Zusätzlich kann der Code reduziert werden auf die zu übenden Bestandteile (z.B. eine Berechnung, Schleife oder Verzweigung). Die Konzentration auf die wesentlichen Code-Bestandteile bietet eine große kognitive Entlastung, die auch in allen Evaluationen immer positiv bewertet wurde. Weiterhin kann jeder Code-Block im Aufgabentext auch direkt ausgeführt werden, wodurch Rückmeldungen und ggf. Hinweise zu Fehlern sofort zur Verfügung stehen.

4 Lessons Learned

Nicht alle Erfahrungen sind positiv, nicht alle sind überraschend, sie führten aber immer dazu, Konzepte und Umsetzungen nach jeder Iteration zu überdenken.

4.1 Flexibilität erfordert zusätzliche Orientierung

Das prinzipiell einfache und positive Konzept des Flow-Programming wurde erst mit der Einführung der Verwendung von Klassen schwierig, da dann Lösungs-Code sowohl im Code-Block im Aufgabentext, als auch im regulären Java-File stehen kann. Positiv daran ist natürlich, dass man Code immer im aktuell gelesenen File schreiben kann und man nicht ständig zwischen Aufgaben-File und Lösungs-Code-File hin und her wechseln muss. Es entstand aber mitunter Verwirrung, wo denn nun der „eigentliche“ Code stand, da es normalerweise genau ein File für den Lösungs-Code gibt.

Flexibilität muss also noch ausführlicher erläutert werden. Die Studierenden müssen in die Lage versetzt werden, sich ein mentales Modell von der Funktionsweise zu machen [Lo16]. Ggf. ist auch der Rückbau von „gut gemeinten“ Features zu erwägen. In jedem Fall müssen die Orientierung und der Zusammenhang immer gewährleistet werden.

Der später im Studium notwendige Übergang von der Nutzung der speziellen IDE 5Code zu einer üblichen IDE muss ebenfalls beachtet werden. Andernfalls gehen die im besten Fall gut verinnerlichten Schritte LVÜAC in den rein Code-orientierten IDEs wieder unter.

4.2 AnfängerInnen sind anders anders

Eine wichtige Beobachtung ist, dass Studierende immer auf ganz eigene Art und Weise Aufgaben bearbeiten. Vor allem die zentrale Zielgruppe der ProgrammieranfängerInnen hat nicht immer die eigens für sie erdachten Hilfsmittel genutzt. So wurden beispielsweise der Code-Explorer (siehe Abb. 3) und selbst die Text-Suche selten genutzt. Das kann mit der Unerfahrenheit und Unsicherheit der Anfänger erklärt werden, die bei (der ja eigentlich zu vermeidenden) kognitiver Überforderung auf bekannte Aktivitäten zurückfallen.

Features müssen daher auf geeignete Art und Weise AnfängerInnen nahegebracht werden. Vielleicht sollten neue Features und Konzepte häufiger in die inhaltliche Aufgabenstellung mit einbezogen werden, um ihre entlastende Funktion laufend sichtbar zu machen.

4.3 Nicht-AnfängerInnen wollen ernst genommen werden

Die Gruppe unserer Studierenden im ersten Semester ist recht heterogen. Die Spanne reicht von völligen AnfängerInnen (ca. 45%), die noch nie irgendetwas programmiert haben, bis hin zu denen, die bereits aus der Schule, Ausbildung oder Studium viel Erfahrungen mit „Programmieren“ haben, d.h. auch mit üblichen IDEs (ca. 35%).

Wer sich bereits für kompetent hält, akzeptiert die didaktischen Konzepte weniger. Studierende mit Vorwissen und Erfahrungen mit anderen IDEs müssen daher passend „abgeholt“ werden. Anfängern und Erfahreneren gleichermaßen gerecht zu werden, ist eine Herausforderung, die wir in der weiteren Entwicklung angehen wollen, z.B. durch Auto-Adaption oder Adaptierbarkeit der IDE oder auch Gamification-Ansätze („Freischalten“).

4.4 Medienkompetenz und Selbständigkeit scheinen eher abzunehmen

Die Beobachtungen der BetreuerInnen im Praktikum deuten übereinstimmend darauf hin, dass über die letzten zehn Jahre hinweg Studierende immer leichter überfordert werden, sowohl vom Inhalt als auch vom technischen Umgang mit einem Programm. Die Selbständigkeit bei der Lösungsfindung scheint auch abzunehmen. Immer mehr scheinen kaum Erfahrung in der Benutzung von Software, des Betriebssystems und der Verwaltung von Dateien zu haben, die über das Browsen des WWW und etwas Schreiben hinausgeht. Die Funktionalität wurde insgesamt in 5Code schon gegenüber professionellen IDEs wie VS Code, eclipse, oder IntelliJ, stark reduziert – aber vielleicht noch nicht genügend.

Inhalte, die Art und Weise der Vermittlung und auch die verwendeten Tools müssen daher weiter an Fähigkeiten der Studierenden angepasst werden sollten. Auch sollten nicht zu viele grundlegende Fertigkeiten im Umgang mit Software vorausgesetzt werden.

Funktionalitäten sollten für ProgrammieranfängerInnen schrittweise eingeführt werden sollten, um die Komplexität der IDE und die kognitive Belastung minimal zu halten. Features sollten nur schrittweise freigeschaltet, d.h. überhaupt zugänglich gemacht, werden.

5 Ausblick

Aufbauend auf den geschilderten Erfahrungen wollen wir in den folgenden Semestern einiges weiter verbessern, z.B.: detailliertere Rückmeldungen bei der Fehlerbehebung und zum Code-Aufbau, Visualisierung des Ablaufs eines Programms und seiner Effekte, Visualisierung des Lernfortschritts, verbunden mit Elementen der Gamification.

Die AutorInnen bedanken sich für die Förderung als Innovation in der Digitalen Hochschullehre in der Gemeinsamen Programmlinie von MIWF/NRW und Stifterverband.

Literaturverzeichnis

- [An12] Anderson, T., Shattuck, J.: Design-based research: A decade of progress in education research? *Educational Researcher*, 41(1), S. 16-25, 2012.
- [CS96] Chandler, P., Sweller, J.: Cognitive load while learning to use a computer program. *Applied Cognitive Psychology*. 10, S. 151-170, 1996.
- [Da05] Dahm, M., *Grundlagen der Mensch-Computer-Interaktion*, Pearson, München, 2005.
- [Da15] Dahm, M., Barnjak, F., Heilemann M., *5Code - Eine integrierte Entwicklungsumgebung für Programmieranfänger*, In: Hans Pongratz, Reinhard Keil (Hrsg.): *Die 13. E-Learning Fachtagung Informatik (Delfi)*, LNI, Seite 119-130, Gesellschaft für Informatik, 2015.
- [Da16] Dahm, M., Barnjak, F., Heilemann M., *5Code – An Integrated Programming Environment for Beginners*. *i-com J of Interactive Media*, Ausgabe 2-2016, DeGruyter, 2016.
- [FV10] Ford, M, Veneme, S, *Assessing the Success of an Introductory Programming Course*, *Journal of Information Technology Education*, Volume 9, 2010.
- [Hu07] Hubwieser, P., *Didaktik der Informatik*, Springer, Berlin, 2007.
- [Ki17] Kirchhof, D., *5Code.Local- Eine integrierte Entwicklungsumgebung für Programmieranfänger*, Hochschule Düsseldorf, FB Medien, 2017.
- [Kn84] Knuth, D. E., *Literate Programming*. Leland Stanford Junior University, United States: Center for the Study of Language and Information, 1984.
- [Kö19] Köhler, M., *Optimierung der Navigation der Entwicklungsumgebung 5Code*, Hochschule Düsseldorf, FB Medien, 2019.
- [Lo16] Loksa, D. Ko, A. J., Jernigan, W., Oleson, A., Mendez, C. J., and Burnett., M. M., *Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance*. In *Proceedings of the 2016 CHI ACM.*, New York, NY, USA, S. 1449–1461, 2016.
- [Ro20] Rose, J.: *Optimierung des Workflows der Programmierungsumgebung 5Code für Programmieranfänger*, Hochschule Düsseldorf, FB Medien, 2020.
- [Wi14] Williams, J.S: *A Computer Learning Environment for Novice Java Programmers That Supports Cognitive Load Reducing Adaptations and Dynamic Visualizations of Computer Memory*, Dissertation, Paper 574, Univ. of Wisconsin, Milwaukee, 2014.