# 5Code - Eine integrierte Entwicklungsumgebung für Programmieranfänger

Markus Dahm[1], Frano Barnjak[2] und Moritz Heilemann[3]

**Abstract:** Aufbauend auf langjähriger Erfahrung in der Informatik-Lehre ist die integrierte Entwicklungsumgebung (IDE) 5Code entstanden, die speziell Programmieranfänger unterstützen soll. Zunächst wurde eine einfach verständliche Darstellung erarbeitet, wie man in fünf Schritten *vom Problem zum Programm* kommt: *Lesen* > *Verstehen* > *Überlegen* > *Aufschreiben* > *Codieren*. Um die kognitive Belastung der Lernenden dabei 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. Zur Unterstützung des Verstehens können Aufgabenteile markiert und mit eigenen Überlegungen annotiert werden. Diese Notizen können in den Code übernommen und synchronisiert werden. 5Code ist als Web-Applikation implementiert und wurde in einem Hochschul-Programmierpraktikum evaluiert.

**Keywords:**

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

## 1 Einleitung und Motivation

### 1.1 Vom Problem zum Programm

Dass es für die meisten Anfänger größere Probleme gibt als nur die Syntax und Semantik von Programmiersprachen, ist eine allgemeine Erfahrung aus dem Unterricht für Programmieranfänger (vgl. [Hu07], [Bo07], [FV10]). Typisch ist eine Aussage wie *„Ich weiß, wie ich eine* `while`*-Schleife schreibe, ich weiß nur nicht, wann ich die hinschreiben soll.“* Die größere Herausforderung, das wesentlich Neue, was verstanden, verinnerlicht und angewendet werden muss, ist die Vorgehensweise, die *vom Problem zum Programm* ([Hu07a]) führt. Im Großen entspricht das der Disziplin des Software-Engineering, für Anfänger muss jedoch eine Darstellung verwendet werden, die nicht selber neue Fragen aufwirft und von den konkreten Problemen des Programmierens im Kleinen ablenkt.

Diese Schwierigkeiten der Lernenden tauchen bereits auf, wenn es sich „nur“ um die Umsetzung prozeduraler Inhalte handelt. Die hier getroffenen Überlegungen sind aber

[1] HS Düsseldorf, FB Medien, Josef-Gockeln-Str. 9, 40474 Düsseldorf, markus.dahm@hs-duesseldorf.de,
[2] HS Düsseldorf, FB Medien, Josef-Gockeln-Str. 9, 40474 Düsseldorf, frano.barnjak@hs-duesseldorf.de
[3] HS Düsseldorf, FB Medien, Josef-Gockeln-Str. 9, 40474 Düsseldorf, moritz.heilemann@hs-duesseldorf.de

auch gültig für die Lehre in Objektorientierter Programmierung (OOP). Die zusätzlichen Probleme, die das Verständnis von OOP bereitet, wie sie z.B. in [Bo07], [Hu07], [Wi14] geschildert werden, addieren sich zu den grundlegenderen Hürden wie man allgemein vom Problem zum Programm kommt. Anders als z.B. in [Wi14] oder [Bo07] wurden daher OOP-Konzepte für die Entwicklung von 5Code noch nicht in den Fokus gestellt[4].

### 1.2 Kognitive Belastung

Die kognitive Belastung nach der Cognitive Load Theory von Sweller und Chandler [CS96] bei der Lösung von Aufgaben ist für Programmieranfänger sehr hoch (vgl. auch [Bo07], [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. Das ist eine zu hohe kognitive Belastung für fast alle Programmieranfänger, da sie sich, anders als Experten, noch nicht auf entsprechend gefestigte Schemata und Mentale Modelle im Langzeitgedächtnis stützen und erlernte Pläne (vgl. [Bo07], [SH11]) zur Problemlösung anwenden können.

### 1.3 Kontext geht häufig verloren

Diese kognitive Überlastung führt dazu, dass man in Programmierpraktika immer wieder beobachten kann, dass sich nach dem ersten, oft nur oberflächlichen, Lesen des Aufgabentextes die Teilnehmer sofort auf die Codierung (als den vermeintlich wichtigsten Teil der Leistung) stürzen – und auch in diesem engen Kontext verbleiben. Durch die Konzentration auf die Umsetzung in Code gerät die eigentliche Aufgabenstellung aus dem Arbeitsgedächtnis und damit aus dem kognitiven Kontext. Im besten Fall wird dann noch einmal nachgesehen, in nicht seltenen Fällen wird irgendetwas codiert, das ungefähr zur Aufgabenstellung oder zum gerade geübten Thema passt. Viele verlieren sich in ihrem Code und können keine Verbindung mehr zur Aufgabe herstellen. Eine Vorstellung, wie sie weiter in Richtung der Lösung kämen, wird damit immer schwerer zu entwickeln, die Unsicherheit, wird immer größer. Diese allgemeinen Defizite bei der Methodik des Problemlösens hat auch schon [MC01] beschrieben.

Diese Situation wird auf der technischen Seite dadurch weiter verschärft, dass jede Entwicklungsumgebung (IDE) nur Code darstellt – als Programmtext oder als logische Blöcke. Die Aufgabenstellung ist nie Teil einer IDE, so muss sie in einem separaten Programm geöffnet werden oder auf zusätzlichem Papier mitgeführt werden. Das gilt auch für Lernumgebungen für Programmieranfänger.

[4] Für die Vermittlung der grundlegenden OO-Konzepte wurden andere Darstellungen und Visualisierungen (z.B. DOOM-Prinzip ([Da12])) entwickelt, die hier aber nicht weiter dargestellt werden können.

# 2 Bestehende Angebote für Programmieranfänger

Lernumgebungen für Programmieranfänger lassen sich gut in Microwelten und Programmierumgebungen einteilen. Einen guten Überblick bietet [Wi15].

**Microwelten** gehen auf das Logo-System von Papert zurück. Innerhalb einer Microwelt formuliert man (einfache) Code-Befehle in einer eigenen Scriptsprache (Logo, Scratch), oder einer „richtigen" Programmiersprache (Squeak, Javahamster, Greenfoot). Microwelten sind für jüngere Programmieranfänger ab ca. zehn Jahren gedacht und wollen die Programmierung attraktiv gestalten, einerseits durch die optische Gestaltung und andererseits durch möglichst niedrigschwelliges Codieren. Der Code wird dann von einer Schildkröte (Logo), einem Käfer (Kara) oder einer Figur (Scratch, Squeak) visuell umgesetzt oder steuert sogar Hardware (Lego). Programmiert wird entweder durch das Zusammenfügen von Puzzleteilen (Scratch, Lego) oder Schreiben von Code.

Die Unterstützung der Microwelten beim Erstellen von Anwendungen und dem damit verbundenen Lösen von Problemen beschränkt sich immer auf das Codieren und schnelle Testen. Als Vorgehen wird dadurch stark trial and error forciert. Im besten Fall wird noch eine fruchtbare Kommunikation innerhalb der Community initiiert, die häufig direkt in der Lernumgebung angesprochen werden kann.

**Programmierumgebungen für Anfänger** ermöglichen das Programmieren in einer verbreiteten Programmiersprache, sehr häufig in Java, aber auch Smalltalk (Squeak). Ihr Ziel ist, sowohl das Erstellen von Code zu vereinfachen als schnell Code ausprobieren zu können. Zielgruppen sind ältere Schüler und auch Studierende an Hochschulen.

In der Lehre von Objektorientierung mit Java wird verbreitet BlueJ eingesetzt. BlueJ ist einfacher als eine vollständige IDE (z.B. eclipse) durch ein zu Beginn eingeschränktes Funktionangebot und bietet zusätzlich ein Klassendiagramm. Das Ausprobieren von Code wird besonders unterstützt durch interaktives Inspizieren von Objekten und Ausführen von Methoden. Greenfoot basiert auf BlueJ und bietet dazu eine Microwelt.

Anfänger-Programmierumgebungen bieten gute Unterstützung für die Umsetzung in Code sowie das interaktive Testen von Code. Alle notwendigen vorangehenden Schritte wie Analyse der Aufgabe, Überlegung und Design der Lösung werden nicht unterstützt. Stattdessen wird ebenfalls trial and error vereinfacht und damit nahegelegt.

Einen aktuellen Trend bilden **webbasierte Online-Kurse** („MOOCs"). Populär sind Lernplattformen wie khanacademy, codeacademy oder code.org. Die Kurse weisen häufig eine ähnliche Struktur auf (Lernen, Anwenden, Lernen, …). Programmier-anfänger lernen an vorgefertigten Beispielen und ausgearbeiteten Kursinhalten die Grundlagen der Programmierung mit visueller Unterstützung. Da die online-Plattformen typischerweise eine Programmierung nicht über den vorgesehenen Ablauf hinaus ermöglichen, lassen sie praktische Aspekte der Software-Engineerings außen vor (Analyse, Design, Code-Organisation, Debugging). Die Vorgehensweise vom Problem zum Programm wird eher selten thematisiert oder gar praktisch vertieft.

# 3 Didaktische Ziele

Das Ziel des Konzepts von 5Code ist, 1) dass die Studierenden den *Weg vom Problem zum Programm* erlernen und anwenden können: Sie sollen nicht, wie von ihnen vielfach irrtümlich angenommen, so schnell wie möglich irgendwie laufenden Code produzieren, sondern *alle* vorher notwendigen Schritte durchlaufen. Wenn die Aufgaben anspruchsvoller werden, sollen sie so eine zielführende Vorgehensweise kennen und geübt haben.

Um 2) die *kognitive Belastung der Lernenden auf dem Weg zu verkleinern*, wird ihnen zunächst explizit klar gemacht, dass sie beim Lösen von Aufgaben – sowohl im vereinfachten Kontext des Praktikums aber auch in der „realen Arbeitswelt" – *nicht alle Tätigkeiten und Anforderungen gleichzeitig bearbeiten* müssen. Im Gegenteil: Die Trennung in einzelne Phasen oder Schritte entlastet und macht es so möglich, jeden einzelnen Schritt auch erfolgreich zu durchlaufen.

Angelehnt an die im Software-Engineering verwendeten Phasen, bzw. Aufgabengebiete Analyse, Design, Implementierung (vgl. [Ba09]) wurde eine auch für Anfänger sehr einfach verständliche Darstellung einer Vorgehensweise erarbeitet, in denen man *in fünf Schritten von Problem zum Programm* kommt ([Da11]). Im Gegensatz zu Fach-Termini wie Analyse oder Design, die Anfängern unbekannt sind, wurden sowohl selbsterklärende Begriffe als auch Verbformen gewählt, die die Aktivitäten der Lernenden selber ausdrücken[5]:

***Lesen* > *Verstehen* > *Überlegen* > *Aufschreiben* > *Codieren***

Überraschend ist vielleicht, dass der erste Schritt *Lesen* ist. Die Erfahrung aus vielen Praktika zeigt aber, dass viele Studierende die vorgegebenen Aufgaben nur überfliegen, nicht komplett oder nicht zu Ende lesen[6]. Dem soll mit der expliziten Nennung vorgebeugt werden. Außerdem wird auf die Reihenfolge der Schritte *Verstehen* und *Überlegen* hingewiesen. Mittels *Aufschreiben* soll gelernt werden, Gedanken zu verbalisieren; und das zunächst in Deutsch und noch nicht in der Programmiersprache. So wird die Semantik der Lösung von ihrer Syntax getrennt; man lernt so, zuerst das Soll zu beschreiben und dann das Aufgeschriebene erst im allerletzten Schritt zu *Codieren*.

All dieses ist für einen Experten selbstverständlich. Anfänger können sich jedoch nicht auf diese Schemata und Pläne stützen (siehe z.B. [SH11]) und bedürfen daher dieser expliziten und wiederholten Regeln um sie durch wiederholte Anwendung zu erlernen[7].

Schließlich soll 3) der *komplette Kontext: Aufgabe + Analyse + Design + Implementierung* gleichzeitig verfügbar sein: Die fünf Schritte sollen nicht isoliert sondern aufeinander folgend und aufeinander aufbauend durchgeführt werden können. Das soll so wenig kognitiven und organisatorischen Aufwand wie möglich erfordern.

[5] Zusätzlich tragen Dozent und BetreuerInnen T-Shirts, auf denen diese Schritte aufgedruckt sind.
[6] Das kann auf Probleme mit der Leistung des Arbeitsgedächtnisses zurückzuführen sein (vgl. [GM10], S. 186)
[7] Siehe dazu auch das Drei-Phasen-Modell von Fitts, z.B. in [GM10], S. 148f

# 4 5Code – die IDE mit komplettem Kontext

Gerade regelorientierte Anfänger stützen sich stark auf die verwendeten Tools (vgl. [Da05], S. 81). Wie oben gezeigt, wird unglücklicherweise von den häufig in praktischen Übungen verwendeten Tools lediglich das Codieren und ggf. das Testen unterstützt. Die Schritte davor werden gar nicht angeboten. Es bedarf also einer eigenen Initiative der Lernenden, die Aufgabe parallel vorzuhalten und zusätzlich eigene Notizen für das eigene Verständnis und für das Planen der Lösung anzufertigen. Beides erfordert sowohl selbstständiges Mitdenken als auch zusätzlichen methodischen und kognitiven Aufwand, da mehrere Tools oder mehrere Medien (Computer, Papier) eingesetzt werden müssen.

## 4.1 Struktur und Vorgehensweise

Um diesen zusätzlichen Aufwand zu vermeiden und so mehr kognitive Kapazität zur Lösung und zum Lernen frei zu machen, bietet 5Code dem Nutzer dafür den gesamten Kontext aller 5 Schritte in drei Fenstern als Teile eines einzigen Browserfensters (Abb1).

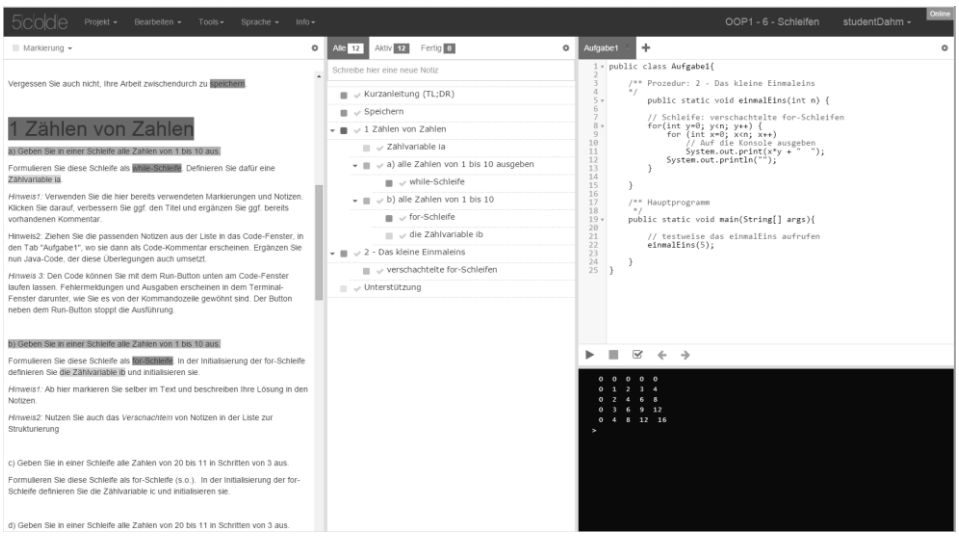


Abb1: Lesen > Verstehen > Überlegen > Aufschreiben > Codieren

Lernende brauchen so keine zusätzlichen Programmfenster, Papiere oder Dateien für die Aufgabenstellung und selbst geschriebenen Überlegungen sondern haben immer alle Bestandteile gleichzeitig präsent. Der gesamte Kontext der Bearbeitung bleibt so immer erhalten. Bei der Lösung einer Aufgabe arbeitet man sich von links nach rechts vor.

### 4.2 Lesen und Verstehen

Im linken Fenster wird die Aufgabe darstellt, wo sie jederzeit gelesen werden kann. Da sie nicht in einem anderen Programmfenster verschwindet, wenn Notizen geschrieben oder wenn codiert wird, ist die Aufgabe auch während des gesamten Bearbeitungsprozesses immer präsent - der komplette Kontext bleibt in 5Code immer erhalten – das Arbeitsgedächtnis wird damit entlastet.

Zur Unterstützung des Verstehens kann man, wie in jedem Texteditor üblich, beliebige Textteile selektieren. Solange das Textstück selektiert ist, kann man die Selektion auch mit einem Typ versehen und so daraus eine Markierung machen. Als Typen stehen zur Auswahl: *Marker* (Marker 1-5), die wie ein Textmarker lediglich in 5 Farben hervorhebt, ohne weitere intrinsische Bedeutung. Um beim Lesen und Verstehen sich selber schon einen Hinweis zu geben, welche Rolle der selektierte Textteil haben könnte, gibt es hierarchisch organisierte Implementierungs-orientierte Typen, die jeweils eine eigene Farbe haben: *Variable* (Parameter, Local Var., Instance Var., Class Var.), *Verzweigung* (if, if-else, switch-case, :?-Operator), *Schleife* (while, do-until, for, for each), *Methode* (Prozedur, Funktion) und *Klasse* (Subklasse, Interface). Der Typ einer Markierung kann nachträglich jederzeit geändert werden. Die Phase des Erarbeitens des Aufgabentextes wird damit ähnlich wie bei der Arbeit mit Papier und Textmarker unterstützt – mit dem Zusatznutzen, dass Markierungen editiert und außerdem in Notizen annotiert werden können (s. Kap. 4.3).

### 4.3 Überlegen und Aufschreiben

Eigene Überlegungen können wie üblich direkt im Code-Fenster als Kommentare geschrieben werden. Allerdings sollten Lernende so lange wie möglich vermeiden zu Codieren – um zu lernen, die notwendigen, vorbereitenden Schritte vor dem Codieren nicht zu überspringen. Daher gibt es die Möglichkeit, in einem eigenen Fenster Notizen anzulegen, dazu kann man Text am Kopf der Notizliste eingeben. Ganz schnell entsteht eine Notiz außerdem direkt aus einer Markierung im Aufgabentext.

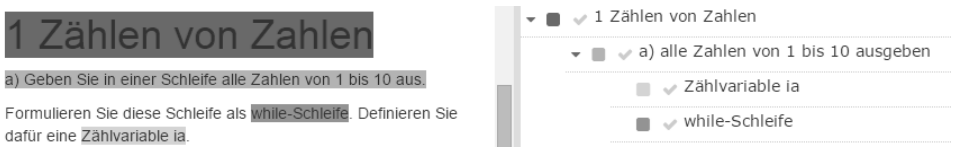


Abb2: Markierung im Aufgabentext und damit verbundene, verschachtelte Notizen

Der markierte Text wird dabei automatisch als Titel der Notiz übernommen. Den Titel kann man dann editieren und mit eigenem Kommentar ergänzen. Dazu klickt man auf die Markierung oder auf die Notiz, was einen entsprechenden Dialog öffnet.

Notizen kann man auch durch einfaches drag&drop sowohl vertikal verschieben als auch hierarchisch anordnen und so logisch einander zuordnen. So kann man z.B. zuerst

Variablen, Schleifen und andere Anforderungen aus dem Aufgabentext einzeln mit dem passenden Typ markieren. Danach ordnet man die daraus entstehenden Notizen einer Notiz einer Funktion zu, mit der diese Anforderungen implementiert werden sollen. Die eigene Anordnung der Notizen entspricht dann der eigenen logischen Ordnung wie man sie sich erarbeitet hat, unabhängig vom Aufbau der Aufgabe (Abb2).

Um bei umfangreicheren Notizlisten nicht die Übersicht zu verlieren, können verschachtelte Notizen ein- bzw. ausgeklappt werden (*folding*), wie man es aus Filesystem-Explorern aus dem Betriebssystem oder von Musiksoftware kennt.

Seine Notizen kann man auch als ToDo-Liste nutzen. Wenn eine Aufgabe, die man durch eine Notiz definiert hat, fertig bearbeitet hat, kann man sie in der Liste abhaken; die Notiz erscheint dann ausgegraut. Die Notizliste kann man außerdem nach fertigen oder nach noch zu bearbeitenden Notizen filtern, so sieht man, was fertig ist und was noch getan werden muss.

### 4.4 Codieren – Tippen, Draggen, Testen und Roundtrip Engineering

So vorbereitet kann man nun (endlich) die Lösung codieren. Der Code-Teil von 5Code unterstützt aktuell Java, ist aber sprachunabhängig und wird zurzeit für HTML/CSS/ Javascript angepasst. Der Editor bietet Komfortfeatures wie Syntax-Highlighting, automatisches Einrücken, Folding, Zeilennummern und Tabs für die zu schreibenden Klassen. Ganz bewusst nicht geboten werden Autocompletion, Codegenerierung und Autocorrection, die für Experten wertvolle Hilfsmittel sind, da sie einfache Tätigkeiten automatisieren. Programmieranfänger müssen diese Routinetätigkeiten aber erst erlernen und verstehen; sie sollen ihnen daher nicht durch ein Tool abgenommen werden. Das Erlernen wird wesentlich durch die eigenen Überlegungen und Handlungen sowie die Korrektur der dabei gemachten Fehler bestimmt (vgl. [GM10]). Diese müssen die Lernenden daher selber durchführen.

5Code kann immerhin das (den meisten Lernenden sehr lästige[8]) Tippen von Kommentaren abnehmen. Wer systematisch arbeitet, im Aufgabentext Teile markiert und die damit erzeugten Notizen mit eigenen Beschreibungen ergänzt hat, der wird dafür dadurch belohnt, dass er seine Notizen direkt in den Code übernehmen kann. Jede Notiz kann einfach durch drag&drop in das Code-Fenster an die gewünschte Stelle gezogen werden: Titel und Kommentar werden als Zeilenend-Kommentar (`//`) in den Code eingefügt, bei Notizen vom Typ Methode oder Klasse wird javadoc-Kommentar (`/** */`) eingefügt.

Der Entwicklungsprozess ist auch bei kleineren Aufgaben häufig iterativ. Beispielsweise fällt erst während der Codierung häufig auf, dass Angaben und Beschreibungen in Kommentaren fehlen oder korrigiert werden müssen. 5Code hilft, Notizen und Code

[8] Das Tippen der Kommentare ist vielen u.a. deswegen lästig, weil die Nutzung der Tastatur eine ablenkende mentale Belastung darstellt. Dem könnte dadurch abgeholfen werden, indem Blindschreiben erlernt wird.

Kommentare konsistent zu halten: Jede Änderung einer Notiz in der Liste oder im Dialog wird im daraus entstandenen Kommentar im Code-Editor sofort automatisch synchronisiert. Ebenso finden sich alle Änderungen eines Kommentars im Code in der jeweiligen Notiz wieder (Abb3). Dieses *Roundtrip Engineering* in 5Code vermindert sowohl den kognitiven als auch den mechanischen Aufwand der mehrfachen Eingabe. Auch wird die Verbindung zwischen Überlegungen und Code damit noch deutlicher.

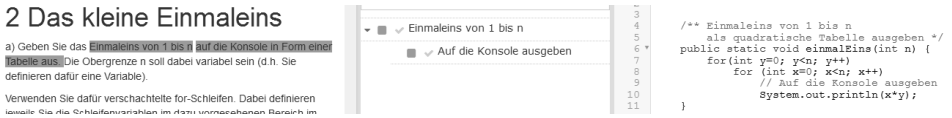


Abb3: Markierung -> Notiz <- *synchronisiert mit* -> Code-Kommentar

Java-Programme können über Buttons compiliert, gestartet und gestoppt werden. Text-Ausgaben auf `System.out` und `System.err` werden auf dem Terminalfenster unterhalb des Codes dargestellt; von dort können auch Tastatur-Eingaben über `System.in` eingelesen werden. Fehlermeldungen des Compilers erscheinen im Terminalfenster, zusätzlich werden sie im Code durch Marker/tooltip neben der Zeilennummer angezeigt.

### 4.5 Vorbereitung und Bewertung durch Lehrende

Lehrende haben in 5Code einige Möglichkeiten zur Vorbereitung: Aufgaben erstellt und editiert man mit gängigen WYSIWYG-Funktionen zur Formatierung, Bilder oder Links können eingebunden werden. Wenn Aufgaben bereits als Text vorliegen, kann man sie mit copy&paste formatiert übernehmen.

Wie [KS06] deutlich zeigen, ist es gerade bei Programmieranfängern sinnvoll, sie nicht mit einer Textaufgabe alleine zu lassen, sondern sie mit zusätzlichen Hinweisen auf „den rechten Pfad“ zu führen. In 5Code müssen diese nicht in der Aufgabe enthalten sein, wodurch der reine Aufgabentext schnell unübersichtlich würde. Mit Markierungen im Aufgabentext kann man Hinweise geben, diese kann man in den zugeordneten Notizen durch Hilfestellungen in deren Kommentaren noch ergänzen.

Außerdem kann man Code vorgeben um Lernenden Tipparbeit, die nicht zum Thema gehört und sie daher nur ablenken würde, abzunehmen. Dazu gehört z.B. auch der Klassenrahmen, der für prozedurale Programminhalte noch gar nicht erklärt werden kann, für den Java-Compiler aber vorhanden sein muss. Analog können für OO-Aufgaben Hilfsklassen in eigenen Tabs mit Code zur Verfügung gestellt werden.

Lehrende können markierte Aufgaben, Notizen und Code der Lernenden einsehen. Eine Online-Abgabe, -Beurteilung und auch -Korrektur wird so möglich.

# 5 Implementierung als Web-Applikation

5Code wurde als Web-Applikation implementiert, was viele Vorteile hat: 5Code kann auf jeder Plattform (Windows, MacOS, Linux) genutzt werden. Es werden keine großen Performance-Anforderungen gestellt. Außerdem können mobile und hybride Geräte wie Tablets ebenfalls genutzt werden. Eine neue Version von 5Code muss nur auf dem Server installiert werden und liegt beim Aufruf im Browser sofort vor. Die Arbeiten der Lernenden liegen immer auf dem Server vor und können dort gesichert werden. Zusätzlich kann jeder Nutzer seine Daten auch lokal abspeichern und auch wieder auf den Server hochladen.

Als Anwendungsserver und Datenserver von 5Code wird Node.js eingesetzt, der im Gegensatz zu reinen Webservern wie Apache weniger aufwändig in der Installation und Administration ist und auch weniger Anforderungen an die Hardware stellt.

Um 5Code zu starten, gibt ein Lernender die Adresse des verwendeten 5Code-*Servers* im Browser an. Von dort wird der Code des 5Code-*Clients* in den Browser geladen und gestartet. Nun läuft 5Code lokal als Javascript-Programm in einem Browserfenster und kann genutzt werden wie ein lokal installiertes Programm, mit ähnlichem User Interface und sehr guter Performance. Dabei gibt es nur eine Einschränkung: Jeder Browser verbietet aus Sicherheitsgründen das Ausführen von Programmen aus einer Webseite heraus. Wenn ein Nutzer seinen Java-Code testen möchte, schickt der Client daher den Java-Code an den Server. Auf dem Server wird der Java-Code zuerst compiliert und dann dort auch ausgeführt. Das Ergebnis wird an den Client zurück geschickt, der dann den Text, bzw. die Fehlermeldung im Terminalfenster darstellt.

Der compilierte Java-Code wird auf dem Server in einer sogenannten Sandbox ausgeführt, die keinen Zugriff auf Systemressourcen wie Files, Netzwerk hat. Dadurch wird sichergestellt, dass Java-Programme von Lernenden den Server nicht kompromittieren können. Damit Lernende auch von zu Hause 5Code nutzen können, muss der Dienst im Internet verfügbar sein. Daher beachtet 5Code etliche Sicherheits-Aspekte: Nur angemeldete Nutzer können 5Code nutzen, die Session der Kommunikation zwischen Client und Server ist geschützt und der Node.js-Server selber läuft mit stark eingeschränkten Rechten.

Benutzer- und Systemdaten legt der Server in Files ab. Die Aufgabenstellung und der Java-Code sind in HTML bzw. Java Format gespeichert, andere Informationen als JSON-Strings in Textdateien. Zusammengehörende Teile wie Aufgaben, Notizen und Java Code werden stets gemeinsam und damit konsistent gespeichert.

Für den sicheren Betrieb muss also nur Node.js und der Code von 5Code installiert werden, keine Datenbank. Der Aufwand und die notwendigen Kenntnisse für Installation und Betrieb von 5Code sind für Administratoren also minimal.

# 6 Evaluation und Weiterentwicklung

5Code wurde im Praktikum der Lehrveranstaltung OOP1 im 1. Semester eines Medieninformatik-Studiengangs mit 90 Teilnehmern durchgehend eingesetzt und evaluiert.

**Direktes Feedback** - Während der Praktikumstermine wurde von den Betreuern auch der Umgang mit 5Code beobachtet wobei sehr unterschiedliche Reaktionen festgestellt wurden: Vor allem die völlig unerfahrenen Studierenden nahmen die integrierte Bearbeitung gerne an. Wer in der Schule oder in der Ausbildung bereits mit „echten" IDEs gearbeitet hatte, verglich 5Code damit und vermisste einige Experten-Features (die bewusst nicht angeboten wurden, siehe Kap. 4.4). Zu Beginn der Nutzung traten unglücklicherweise technische Probleme auf. Diese konnten zwar innerhalb weniger Tage gelöst werden, leider waren dadurch nicht wenige Studierende negativ geprägt.

Die Studierenden wurden außerdem während des Semesters in drei online-Umfragen befragt. Dabei wurden jeweils neun Fragen zu Nutzung und empfundenem Nutzen über surveymonkey gestellt und eine Auswertung mit attrakdiff ([HB08]) durchgeführt.

**Umfrage U1 –** Nach 5 Wochen Arbeit mit Kommandozeile und einfachem Texteditor: Erfahrungen mit IDEs hatten ca. 2/3 der Teilnehmer (eclipse:35%, Visual Studio: 27%), wobei 1/3 nur „ein bisschen" Erfahrungen angaben. Die Hälfte derer mit Vorkenntnissen hatte sie sich selber beigebracht, die andere Hälfte hatte sie in der Schule/Ausbildung erlernt. Knapp 1/3 arbeitete mit einem Papier-Ausdruck der Aufgabe, die anderen direkt im pdf-file. 85% der pdf-Nutzer machten sich dabei so gut wie keine eigenen Notizen, obwohl sowohl Markierungen als auch Notizen in pdf-files möglich sind, sondern codierten direkt. Die Hälfte der Papiernutzer machen sich Markierungen oder Notizen. Die Arbeit mit mehreren Fenstern fand ¼ umständlich. Andererseits fand die Hälfte es wünschenswert, Aufgaben, Notizen und Code digital verknüpfen zu können. Sowohl pragmatische als auch hedonische Qualitäten der sehr einfachen Tools wie Kommandozeile und einfachem Texteditor wurden über attrakdiff neutral bewertet.

**Umfrage U2 –** 6 Wochen nach U1 zu den ersten Erfahrungen mit 5Code: Die attrakdiff-Ergebnisse zeigen eine Verbesserung der hedonischen Qualitäten. Die in den ersten Wochen noch aufgetretenen technischen Schwierigkeiten haben die Bewertung der pragmatischen Qualität negativ beeinflusst. Es verwendeten 30% die Markierungen in der Aufgabe, 26% nutzen auch spezifische Markierungstypen. 38% ergänzten eigene kurze Kommentare, immerhin 25% sogar längere. ¾ gaben an, dass 5Code beim Lesen und Verstehen hilft. Die hierarchische Ordnung wurde von 38%, das Abhaken von ¼ genutzt. 2/3 gucken immer wieder auf die Aufgabe im Fenster. 60% sagen, dass 5Code beim Überlegen und Aufschreiben hilft. 90% geben an, dass 5Code beim Codieren hilft. Ca. ¼ nutzen die Verknüpfung von Notizen und Code und bewerten das als sehr praktisch. 61% würden empfehlen, 5Code weiter zu nutzen, davon 32% auf jeden Fall. Weitere 26% empfehlen 5Code mit starken Einschränkungen, die sich im Wesentlichen auf die Beseitigung der „Kinderkrankheiten" beziehen.

Aufgrund der vielen Anregungen wurden noch während des laufenden Semesters wesentliche Verbesserungen entwickelt und auch schon eingesetzt (Details siehe U3).

**Umfrage U3 –** Nach weiteren 5 Wochen zu abschließenden Erfahrungen mit 5Code: Diese Umfrage war numerisch nicht sinnvoll auszuwerten, da es (in der letzten Semesterwoche) zu wenig Rückläufer gab. Nach Beobachtungen der BetreuerInnen hat es aber viele positive Rückmeldungen gegeben, was auch von den verbesserten Bewertungen der pragmatischen und hedonischen Qualitäten in attrakdiff gestützt wird. Alle anfänglich aufgetretenen Fehler konnten beseitigt werden, was dazu führte, dass nun ¾ empfehlen, 5Code weiterhin einzusetzen. Etwa ¼ der Studierenden blieb kritisch, vor allem, weil wie sich gegängelt fühlten und lieber eine gewohnte IDE verwenden würden – das sind aber weniger als die Hälfte der 2/3 Studierenden mit Vorkenntnissen, die andere Hälfte war zufrieden. Besonders positiv wurde vermerkt, dass das neue Design gut gefällt, dass im Code-Fenster die Farben nun anpassbar sind, was sofort von ca. 1/3 für ein „Nachtdesign“ genutzt wurde, das Folding von Notizen und dass der Umgang mit Fehlern die der Compiler meldet durch die in Kap. 4.4 beschriebenen Marker stark vereinfacht wurde.

# 7 Fazit und Ausblick

Die integrierte Programmierumgebung 5Code unterstützt explizit die fünf Schritte vom Problem zum Programm: *Lesen* > *Verstehen* > *Überlegen* > *Aufschreiben* > *Codieren*. Das wird erreicht durch die Integration des kompletten Kontexts von Aufgabenstellung, Markierungen, eigenen Notizen und Kommentaren, Codierung und Test in einer einzigen Anwendung. Der Kontext bleibt erhalten, die kognitive Belastung wird gesenkt.

Die Implementierung als Web-Applikation erleichtert in Schulen und Hochschulen die Verwaltung von Applikation, Aufgaben und Lösungen. Zudem kann 5Code so unabhängig von den Hard- und Software der Lernenden eingesetzt werden.

5Code wurde in einem Anfängerpraktikum mit 90 Studierenden eingesetzt und evaluiert. Während des laufenden Semesters konnten bereits einige wichtige Anregungen aus dem Feedback umgesetzt werden. Es konnten Verbesserungen im Design und der Funktionalität erreicht werden, was zu guter Akzeptanz und hohem Nutzen geführt hat.

Zurzeit werden weitere Features eingebaut, unter anderem die Möglichkeit zur Offline-Arbeit, Unterstützung der Arbeit mit Tablet-Computern sowie Mandantenfähigkeit. Außerdem wird 5Code erweitert, um nicht nur in der Lehre in OOP mit Java sondern auch in der Webprogrammierung mit HTML/CSS/Javascript im Praktikum eingesetzt zu werden. Weitere Lehrgebiete in denen Aufgabentexte analysiert werden, auch wenn sie keinen Code sondern Text als Arbeitsergebnis haben, z.B. Deutsch, Englisch, Geschichte, können ebenfalls vom integrierten Ansatz von 5Code profitieren.

## Literaturverzeichnis


[Ba09] Balzert, H. Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering, Spektrum Akademischer Verlag, Heidelberg, 2009

[Bl76] Bloom, B., S., Taxonomie von Lernzielen im kognitiven Bereich. Beltz Verlag, Weinheim 1976

[Bo07] Börstler, J., Objektorientiertes Programmieren – machen wir irgendwas falsch?, In: Sigrid Schubert (Hrsg.), Didaktik der Informatik in Theorie und Praxis, INFOS 2007, (LNI), P-112, Köllen Verlag, Bonn, 2007

[CS96] Chandler, P., Sweller, J.: Cognitive load while learning to use a computer program. Applied Cognitive Psychology. 10, 1996, S. 151-170

[Da05] Dahm, M., Grundlagen der Mensch-Computer-Interaktion, Pearson, München, 2005

[Da11] Dahm, M., Skript zur Vorlesung „Objektorientierte Programmierung 1 und 2“, HS Düsseldorf, 2011

[FV10] Ford, M, Veneme, S, Assessing the Success of an Introductory Programming Course, Journal of Information Technology Education, Volume 9, 2010

[HB08] Hassenzahl, M., Burmester, M., & Koller, F. AttrakDiff: Ein Fragebogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität, In: Ziegler, J. & Szwillus, G. (Hrsg.), Mensch & Computer 2003, S. 187-196, B.G. Teubner, Leipzig, 2003

[Hu07] Hubwieser, P.: A smooth way towards object oriented programming in secondary schools. In: Benzie D., Iding M. (Hrsg.): Informatics, Mathematics and ICT: a 'golden triangle'. IFIP WG 3.1 & 3.5 Working Conference CD proceedings, IFIP & College of Computer and Information Science, NE University Boston, Mass., USA, 2007, S. 1-11

[Hu07a] Hubwieser, P., Didaktik der Informatik, Springer, Berlin, 2007

[KS06] Kirschner, P., Sweller, J, Clark, R, Why Minimal Guidance During Instruction Does Not Work, EDUCATIONAL PSYCHOLOGIST, 41 (2), S. 75–86, Taylor & Francis, USA, 2006

[GM10] Gluck, M., Mercado, E., Myers, C., Lernen und Gedächtnis: Vom Gehirn zum Verhalten, Spektrum Akademischer Verlag, Heidelberg, 2010

[MC01] McCracken, M., Kolikant, Y., Almstrum, V., Laxer, C., Diaz, D., Thomas, L., et al. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students, ACM SIGCSE, Bulletin 33 (4), S. 125-140

[SH11] Saifoulline, P; Hemberger, Ch, Kognitive Kernkompetenzen zum Aufbau fundierter mentaler Modelle für die Bearbeitung komplexer Planungsprobleme, Journal Psychologie des Alltagshandelns / Psychology of Everyday Activity, Vol. 4 / No. 2, university press, Innsbruck 2011, S. 31-44

[Wi14] Williams, J.S:, A Computer Learning Environment for Novice Programmers That Supports Cognitive Load Reducing Adaptations an Dynamic Visualisations of Computer Memory, Dissertation, Paper 574, Univ. of Wisconsin, Milwaukee, 2014

[Wi15] http://de.wikipedia.org/wiki/Erziehungsorientierte_Programmiersprachen, 28.02.2015